Appunti di Sistemi Multiagente

Giovanni Bindi
Università degli Studi di Firenze

Indice

1 Introduzione ai Sistemi Multi-Agente

1.1 Definizioni principali

Partiamo dal concetto di **agente**: per agente si intende qualunque elemento che interagisce con l'ambiente in cui si trova e *prende decisioni* in modo *autonomo*.
L'agente riceve informazioni dall'*ambiente* (per esempio attraverso dei sensori) ed agisce su di esso mediante degli attuatori. Questo paradigma è del tutto generale e gli agenti possono anche essere completamente virtuali.

L'interazione tra agente e ambiente è descritta da un sistema dinamico:

$$\begin{cases} x(t+1) = f(x(t), u(t)) \\ y(t) = h(x(t)) \end{cases} \tag{1.1.1}$$

in cui $x(t)$ è lo **stato complessivo** al tempo t e $u(t)$ è la **decisione/azione** intrapresa dall'agente al tempo t.
Per *stato* si intende la configurazione in cui si trova il sistema agente/ambiente, inoltre $y(t)$ rappresenta l'informazione (dati) raccolta dall'agente (sempre al tempo t) mediante un qualche processo di misura.

Si ha che $x(t) \in \mathbb{X}$ in cui $\mathbb{X}$ prende il nome di **spazio di stato**. Questo può essere un insieme continuo o discreto. La stessa cosa avviene per le azioni $u(t) \in \mathbb{U}$ in cui $\mathbb{U}$ prende il nome di **spazio delle decisioni/azioni** e anche questo può essere uno spazio continuo o discreto.
In generale questi due insiemi possono anche essere ibridi, si pensi infatti all'azione di guidare una macchina: si ha un'azione definita su uno spazio continuo dato dalla manovrazione del volante e un'azione discreta sul cambio della marcia.

Noi siamo interessati ad ***agenti intelligenti***, ovvero agenti che devono soddisfare certe proprietà:

- *Autonomia*: l'agente deve avere controllo totale sulle proprie decisioni.
- *Utilità*: l'agente agisce in modo da perseguire un certo *obiettivo*.
- *Apprendimento*: l'agente é in grado di *imparare* dal passato.

Definiamo un **vettore informativo**:

$$I(t) \coloneqq \begin{bmatrix} y(0) \\ u(0) \\ y(1) \\ \vdots \\ u(t-1) \\ y(t) \end{bmatrix} \tag{1.1.2}$$

che rappresenta tutta l'informazione disponibile al tempo t. La decisione su quale azione applicare al tempo t dipende quindi da:

- Informazione disponibile $I(t)$.
- Obiettivo da perseguire.

Si ha quindi che l'azione/decisione prende la forma della cosiddetta *legge di controllo dell'agente* (scelta sulla base dell'obiettivo), che si può esprimere matematicamente come

$$u(t) = \gamma(t, I(t)) \tag{1.1.3}$$

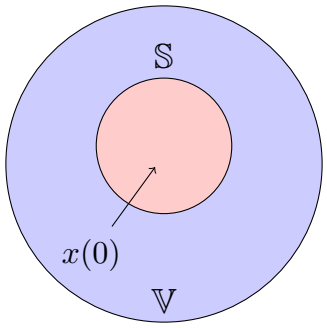
Come si definisce l'obiettivo dell'agente? Possiamo farlo attraverso tre step:

1. La definizione di un insieme $\mathbb{S} \subset \mathbb{X}$ di *configurazioni desiderate*, ovvero un insieme che deve rappresentare l'obiettivo di raggiungimento di una particolare configurazione presente nello spazio di stato (es. vincere una partita di tennis, essere promosso all'esame...).
2. La definizione (se necessaria) anche di un insieme $\mathbb{V} \subseteq \mathbb{X}$ detto *delle configurazioni ammissibili* (o dei vincoli) che il nostro agente deve soddisfare (es. restare all'interno della carreggiata in un sistema di guida autonoma...). Supponendo $x(0) \in \mathbb{V}$ si vuole che $x(t) \in \mathbb{V}, \forall t$.
3. La definizione di una *funzione obiettivo da massimizzare* (o, equivalentemente, di una funzione di costo da minimizzare)
$$r(t, x(t), u(t)) \tag{1.1.4}$$
che rappresenta il **reward** (ricompensa) *istantaneo* che si ottiene al tempo t partendo dallo stato $x(t)$ e applicando l'azione $u(t)$.

L'obiettivo da massimizzare sarà quindi un qualcosa tipo[1]:

$$\sum_{t=0}^{T} r(t, x(t), u(t)) \tag{1.1.5}$$

in cui T prende il nome di orizzonte di controllo (il quale può anche essere ∞: in questo caso per poter definire correttamente questa quantità si dovrà richiedere la convergenza della serie).



Facciamo un po' di **osservazioni**: in questo modello le transizioni sono *deterministiche*. In molti casi (la maggioranza) le transizioni sono incerte e si possono considerare delle variazioni:

[1] Utilizzando una sommatoria il problema diventa trattabile, ma non è l'unica possibilità.

- Modelli con **disturbo**:
$$\begin{cases} x(t+1) = f(x(t), u(t), \xi(t)) \\ y(t) = h(x(t), \eta(t)) \end{cases} \tag{1.1.6}$$
in cui i processi stocastici $\xi(t)$ ed $\eta(t)$ prendono, rispettivamente, il nome di *disturbo di processo* e *rumore di misura* al tempo t.

- Modelli **probabilistici**: si definisce una *densità di probabilità* di transizione:
$$\begin{cases} \varphi(x(t+1)|x(t), u(t)) \\ g(y(t)|x(t)) \end{cases} \tag{1.1.7}$$
in cui $\varphi(\cdot|\cdot)$ rappresenta[2] la probabilità di trovarsi nello stato $x(t+1)$ quando si effettua l'azione $u(t)$ partendo dallo stato $x(t)$ mentre $g(\cdot|\cdot)$ (la funzione di verosimiglianza) rappresenta la probabilità di ricevere l'osservazione $y(t)$ quando lo stato in cui si trova l'agente è $x(t)$.

Inoltre per sistemi con spazio di stato continuo si possono avere modelli di transizione a tempo continuo:
$$\dot{x} = f(x(t), u(t)) \tag{1.1.8}$$

1.2 Sistema Multi-Agente

Definizione 1. *Sistema multi-agente:*Un sistema multi-agente è un sistema in cui $N > 1$ agenti interagiscono tra loro e con l'ambiente prendendo decisioni in modo **autonomo** e **indipendente**. Dal punto di vista matematico si ha:
$$\begin{cases} x(t+1) = f(x(t), u_1(t), \dots, u_N(t)) \\ y_i(t) = h_i(x(t)), \qquad\qquad i \in \{1, \dots, N\} \end{cases} \tag{1.2.1}$$

in cui $u_i(t)$ rappresenta l'azione/decisione dell'agente i al tempo t cosi come $y_i(t)$ l'informazione/dati ricevuti da i al tempo t.

Ogni agente i-esimo ha anche un vettore di informazione disponibile $I_i(t)$ ed ogni agente ha i suoi obiettivi, che possono essere definiti, come prima, in termini di:

- $\mathbb{S}_i \subset \mathbb{X}$: Le configurazioni desiderate dall'agente i.
- $\mathbb{V}_i \subseteq \mathbb{X}$: Le configurazioni ammissibili per l'agente i.
- $r_i(t, x(t), u_1(t), \dots, u_N(t))$: Il reward dell'agente i-esimo (che in generale dipende anche dalle decisioni degli altri agenti, che non conosce): può essere scritto in modo più preciso come $r_i(t, x(t), u_i(t), x(t+1)$, ma la sua definizione dipende dal contesto ogni volta.

La decisione $u_i(t)$ dipende poi dall'informazione locale e dall'obiettivo dell'agente i:
$$u_i(t) = \gamma_i(t, I_i(t)) \tag{1.2.2}$$

[2]Questa prende solitamente il nome di *pdf* di transizione di Markov.

e questo rappresenta un problema di controllo intrinsecamente **distribuito/decentralizzato** dal momento che ogni agente decide in modo **indipendente**.

Tra le tipologie di sistemi multi-agenti possiamo avere:

- Sistemi *cooperativi*, in cui gli agenti hanno gli stessi obiettivi:

$$\mathbb{S}_i = \mathbb{S}, \mathbb{V}_i = \mathbb{V}, r_i = r, \qquad \forall i \tag{1.2.3}$$

- Sistemi *competitivi*, in cui gli agenti hanno obiettivi incompatibili/contrastanti:

$$\mathbb{S}_i \cap \mathbb{S}_j = \emptyset, \qquad \forall i \neq j \tag{1.2.4}$$

- Sistemi *misti*: una tipologia intermedia tra i due estremi di cooperazione totale e competizione totale.

Si può poi definire una classe di sistemi multi-agente ovvero i SMA a **struttura disaccoppiata**: questi sono i sistemi in cui si può decomporre la transizione complessiva 1.2.1 in N transizioni

$$x_i(t+1) = f_i(x_i(t), u_i(t)), \qquad \forall i = 1, \dots, N \tag{1.2.5}$$

ovvero quei sistemi in cui lo stato complessivo si può decomporre in tante parti ed ogni agente agisce solamente su una di esse. Si può poi definire un vettore di stato collettivo:

$$x(t) = \begin{bmatrix} x_1(t) \\ \vdots \\ x_N(t) \end{bmatrix} \tag{1.2.6}$$

in cui l'agente i-esimo agisce solo sul sottostato $x_i(t)$. In questo caso non si ha interazione diretta ma resta un problema multiagente nel senso in cui l'obiettivo dipende dallo stato collettivo. Si ha comunque accoppiamento in termini di:

- Informazione disponibile all'agente i.
- Obiettivo collettivo relativo a $x(t)$.

Tipicamente quindi si suppone che ogni agente i abbia informazioni:

- Sul proprio stato $x_i(t)$.
- Sullo stato di un certo numero $n_i \subseteq \{1, \dots, N\}$ di agenti vicini in cui

$$y_i(t) = \begin{bmatrix} x_i(t) \\ x_j(t), \quad j \in n_i \end{bmatrix} \tag{1.2.7}$$

Esempio 1.1. *Controllo di formazione*

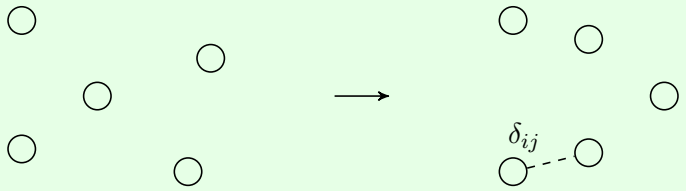
Abbiamo un sistema del tipo

$$\begin{cases} x_i(t) : \text{posizione del robot } i \text{ al tempo } t \\ x_i(t+1) = x_i(t) + T_s u_i(t) \end{cases} \tag{1.2.8}$$

in cui T_s è il tempo di campionamento e $u_i(t)$ la velocità del robot i al tempo t.
In questo caso l'obiettivo può essere portare il sistema multi-robot in una formazione desiderata espressa, ad esempio, in termini di distanze relative desiderate:

$$\mathbb{S} = \{x(t) : ||x_i(t) - x_j(t)|| = \delta_{ij}, \quad \forall i, j\} \tag{1.2.9}$$

in cui δ_{ij} è la distanza desiderata relativa tra i robot i e j.



Tipicamente ogni robot ha informazione:

- Sulla sua posizione $x_i(t)$.
- Sulla posizione dei robot che si trovano a una certa distanza di comunicazione ρ.

In questo caso quindi l'insieme dei vicini (per il robot i) può essere definito come:

$$n_i(t) = \{j \neq i : ||x_i(t) - x_j(t)|| \leq \rho\} \tag{1.2.10}$$

Esempio 1.2. *Calcolo distribuito della media*

Supponiamo di avere N centri di calcolo interconnessi tra loro in qualche modo. Il modello è molto semplice:

$$x_i(t+1) = u_i(t) \tag{1.2.11}$$

In cui $x_i(t)$ rappresenta un certo valore presente in memoria, al tempo t, nel centro di calcolo i mentre $u(t)$ rappresenta il nuovo valore, calcolato al tempo t.
Si parte da uno stato iniziale $\{x_1(0), x_2(0), \ldots, x_N(0)\}$, che può essere ad esempio una misurazione effettuata (separatamente in ogni centro di calcolo) del quale si vuole calcolare la media distribuita. Ogni agente può inoltre comunicare con un insieme di n_i di agenti vicini. L'obiettivo può essere quindi definito come:

$$\mathbb{S} = \{x(t) : x_i(t) = \bar{x}, \quad \forall i\} \tag{1.2.12}$$

in cui $\bar{x}$ rappresenta la media:

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i(0) \tag{1.2.13}$$

2 Elementi di Teoria dei Grafi

2.1 Brevi richiami

In un sistema multiagente si ha scambio di informazione tra agenti (vedi controllo, calcolo distribuito). Emerge naturalmente una struttura a **grafo** in cui

- Gli agenti sono i nodi (o vertici).
- C'è un arco tra i e j quando c'è scambio di informazione: lo scambio per ora viene assunto sempre bidirezionale e quindi si considerano grafi indiretti/non orientati.

Un grafo, indiretto non orientato, si puo esprimere come una coppia

$$\mathcal{G} = (\mathcal{N}, \mathcal{E}) \tag{2.1.1}$$

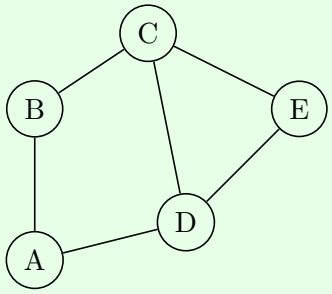
in cui $\mathcal{N} = \{1, 2, \ldots, n\}$ è l'insieme de nodi ed $\mathcal{E}$ l'insieme degli archi.

Un arco è rappresentato da una coppia non ordinata di nodi e si esprime come $\{i, j\}$ dove $\{i, j\} \in \mathcal{E}$ e $i \in \mathcal{N}_i, j \in \mathcal{N}_j$. Gli insiemi $\mathcal{N}_i$ e $\mathcal{N}_j$ rappresentano l'insieme dei **vicini del nodo**, rispettivamente dei nodi i e j.

$$\mathcal{N}_i = \{j \neq i : \{i, j\} \in \mathcal{E}\} \tag{2.1.2}$$

Chiamiamo d_i il grado del nodo i, ovvero il numero di vici del nodo i: $d_i = |\mathcal{N}_i|$.

Esempio 2.1. *Semplice grafo connesso*



In questo caso l'insieme dei nodi è $\mathcal{N} = \{A, B, C, D, E\}$.
L'insieme dei vicini per il nodo A ad esempio è $\mathcal{N}_A = \{B, D\}$ e il suo grado risulta quindi $d_A = 2$.
Si vede facilmente che in questo grafo il grado medio è $\bar{d} = 2.4$.

Una prima misura, molto rudimentale, di connettività del grafo può essere data dal **grado medio**:

$$\bar{d} = \frac{1}{n} \sum_{i=1}^{n} d_i \tag{2.1.3}$$

ma, più in generale, siamo interessati alla **distribuzione dei gradi** P_d (con $d = 1, 2, \ldots, d_{max}$, dove $d_{max} = \max_i d_i$) che rappresenta la *frazione di nodi* del grafo avente grado d. Ovviamente si deve avere che

$$\sum_{d=1}^{d_{max}} P_d = 1 \tag{2.1.4}$$

ed in generale cambia in funzione del tipo di grafo.

In un grafo con connessioni geometriche ad esempio, in cui ogni nodo rappresenta un punto dello spazio con le proprie coordinate (ad esempio un gruppo di sensori), potremmo avere una situazione in cui due nodi sono connessi se e solo se la loro distanza relativa è minore di un certo raggio ρ di comunicazione. In questi casi si ha una distribuzione concentrata intorno al grado medio che, sotto certe ipotesi, si può vedere essere una distribuzione binomiale.

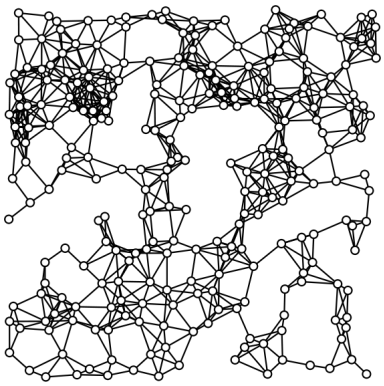


Figura 1: Esempio di un grafo geometrico.

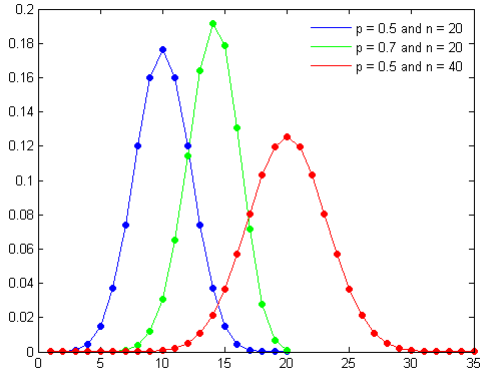


Figura 2: P_d (distribuzione binomiale) al variare dei parametri.

Si possono poi avere distribuzioni di tipo *power law* in cui $P_d = Kd^{-\alpha}$ che sono, in un certo senso, antitetiche rispetto alle prime. In questo caso ci sono tanti nodi con pochi vicini e pochi nodi (detti **hub**) con molti vicini. In casi tipici $\alpha \in (2, 3)$ e prendono il nome di distribuzioni *heavy-tail*.

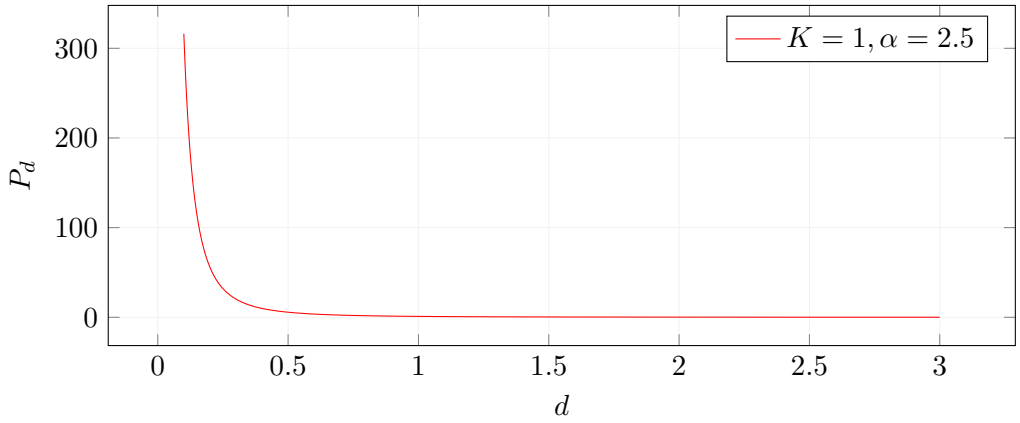


Figura 3: La rappresentazione non è del tutto corretta (K dovrebbe fungere da costante di normalizzazione).

In questo tipo di grafi si ha spesso il fenomeno del preferential attachment, in cui quando un nuovo nodo si aggiunge al grafo è più probabile che si connetta ai nodi hub (quelli che hanno già tante connessioni). Ci possono essere quindi strutture molto diverse in termini di distribuzioni dei gradi.

2.2 Connettività di un grafo

Si definisce la matrice di adiacenza A:

$$A_{ij} = \begin{cases} 1, & \text{se } \{i,j\} \in \mathcal{E} \\ 0, & \text{altrimenti} \end{cases} \tag{2.2.1}$$

che fornisce una descrizione completa del grafo ed è utile per fare i calcoli. Si vede che che ogni riga somma al grado:

$$\sum_j A_{ij} = d_i \tag{2.2.2}$$

e si ha poi che

$$A^2_{ij} = (AA)_{ij} = \sum_{k=1}^{n} A_{ik}A_{kj} \tag{2.2.3}$$

in cui

$$A_{ik}A_{kj} \neq 0 \iff \{i,j\} \text{ sono collegati attraverso } k \tag{2.2.4}$$

da cui si ha che A^2_{ij} rappresenta il numero di **cammini di lunghezza** 2 che connettono i e j e, in generale, A^p_{ij} rappresenta il numero di cammini di lunghezza p che connettono i e j.

Tornando all'esempio 2.1 si vede che

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}, \quad A^2 = \begin{bmatrix} 2 & 0 & 2 & 0 & 1 \\ 0 & 2 & 0 & 2 & 1 \\ 2 & 0 & 3 & 1 & 1 \\ 0 & 2 & 1 & 3 & 1 \\ 1 & 1 & 1 & 1 & 2 \end{bmatrix}, \quad A^3 = \begin{bmatrix} 0 & 4 & 1 & 5 & 2 \\ 4 & 0 & 5 & 1 & 2 \\ 1 & 5 & 2 & 6 & 4 \\ 5 & 1 & 6 & 2 & 4 \\ 2 & 2 & 4 & 4 & 2 \end{bmatrix}$$

Se si analizza A^3_{ii} si può notare poi che questo rappresenta il numero dei *triangoli* nel grafo aventi come vertice i (in realtà sarebbe due volte questo numero dal momento che non è orientato)

Quest'ultima proprietà può essere utile per studiare il **coefficiente di clustering**:

$$\frac{\mathrm{Tr}A^3}{\sum_{i\neq j} A^2_{ii}} = \frac{\sum i = 1 A^3_{ii}}{\sum_{i\neq j} A^2_{ij}} = \frac{6 \times \text{num. traingoli nel grafo}}{\text{num. triangoli potenziali}} \tag{2.2.5}$$

in cui il numero di triangoli potenziali è il numero di coppie di nodi connesse da un cammino di lunghezza 2.

Il coefficiente di clustering rappresenta la tendenza del grafo ad avere una struttura *comunitaria*, ovvero una misura della presenza di cluster (**NB**: un triangolo è una comunità di dimensione 3).

Definizione 2. *Cammino:* Un cammino di lunghezza k tra due nodi i_0, i_k in un grafo indiretto è una sequenza di nodi $c(i_0, i_k) = (i_0, i_1, \ldots, i_k)$ tali che $\{i_{j-1}, i_j\} \in \mathcal{E}$.

Definizione 3. *Grafo connesso:* Un grafo si dice connesso quando, per ogni coppia di nodi i, j esiste un cammino che li collega:

$$\forall i, j \in \mathcal{N} : \exists\ c(i,j) \tag{2.2.6}$$

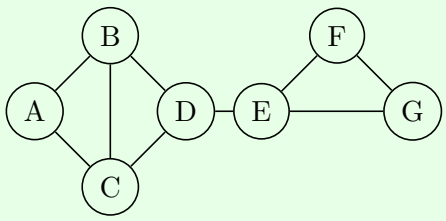
La connettività è quindi una proprietà (binaria) del grafo e ci chiediamo ora se sia possibile averne una misura quantitativa. Esistono tanti modi, ovviamente: un modo per definire una misura del grado di connettività è attraverso la **distanza** δ_{ij}, ovvero la *lunghezza del cammino minimo* tra i e j.
Tra le altre misure che ci possono interessare possiamo poi annoverare

- Distanza *media*:
$$\bar{\delta} = \frac{1}{n(n-1)} \sum_{i \neq j} \delta_{ij} \tag{2.2.7}$$
- Distanza *massima*:
$$\delta_{max} = \max_{ij} \delta_{ij} \tag{2.2.8}$$

Queste misure sono importanti perchè rappresentano, o aiutano a quantificare, la velocità di **diffusione dell'informazione** nel grafo (nel contesto di un SMA). In molti grafi (ad esempio quelli con distribuzioni *power law*, ma non solo) si ha il cosidetto small world effect, ovvero in cui la distanza media $\bar{\delta}$ cresce lentamente (ad esempio come $\log n$) al crescere del numero n di nodi del grafo.

Oltre alla velocità di diffusione siamo anche interessati alla **robustezza** (della connettività rispetto alle disconnessioni) del grafo, che non può essere quantificata nei termini precedentemente introdotti.
Definiamo quindi due quantità:

1. La *node connectivity* $\mathcal{V}_n$, ovvero il minimo numero di nodi che si devono rimuovere per rendere il grafo disconnesso.
2. La *edge connectivity* $\mathcal{V}_e$, ovvero il minimo numero di archi si devono rimuovere per rendere il grafo disconnesso.



In una struttura come questa si ha, ad esempio, che $\mathcal{C}_n = \mathcal{C}_e = 1$ dal momento che basta togliere, rispettivamente, il nodo D (o il nodo E) oppure l'arco tra i due per rendere il grafo disconnesso.

2.3 Laplaciano di un grafo

È uno strumento utile per studiare le proprietà di un grafo.

Definizione 4. *Matrice diagonale dei gradi:* Dato un grafo $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, indiretto non orientato, con $n = |\mathcal{N}|$, definiamo la **matrice diagonale dei gradi**:

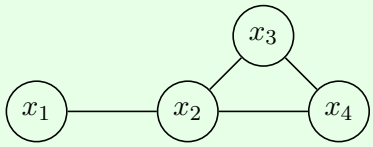
$$D \coloneqq diag\{d_1, d_2, \ldots, d_n\} \tag{2.3.1}$$

Definizione 5. *Laplaciano:* Il **laplaciano** L di un grafo (o matrice laplaciana) è definito come

$$\mathsf{L} \coloneqq D - A \tag{2.3.2}$$

dove A è la matrice di adiacenza (definita in 2.2.1).

Esempio 2.2. *Calcolo del laplaciano di un grafo*



Considerando questo grafo si vede che

$$D = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \text{ e } A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

da cui si ha (notando che la matrice di adiacenza è una matrice **simmetrica per un grafo indiretto**) il laplaciano

$$\mathsf{L} = D - A = \begin{bmatrix} -1 & -1 & 0 & 0 \\ -1 & 3 & -1 & -1 \\ 0 & -1 & 2 & -1 \\ 0 & -1 & -1 & 2 \end{bmatrix}$$

In generale si ha che, definendo $\mathcal{N}_i$ l'insieme dei vicini del nodo i

$$\mathsf{L}_{i,j} \coloneqq \begin{cases} d_i, & j = i \\ -1, & j \in \mathcal{N}_i \\ 0, & \text{altrimenti} \end{cases} \tag{2.3.3}$$

il laplaciano contiene quindi *tutte* le proprietà del grafo, così come la matrice di adiacenza, e il loro uso (del laplaciano o della matrice di adiacenza) dipende dall'applicazione. Elenchiamo alcune delle **proprietà** del Laplaciano:

1. Ogni riga somma a 0.
$$\sum_j \mathsf{L}_{ij} = 0$$
2. È una matrice simmetrica (quando il grafo è indiretto), ovvero $\mathsf{L} = \mathsf{L}^\intercal$.
3. Anche le colonne sommano a 0 (per la simmetria, quindi solo nei grafi indiretti).
$$\sum_i \mathsf{L}_{ij} = 0$$
4. È una matrice diagonale dominante.
$$\mathsf{L}_{ii} = \sum_{i \neq j} |L_{ij}|$$
5. È una matrice semidefinita positiva.
$$x^\intercal \mathsf{L} x \geq 0, \quad \forall x \in \mathbb{R}^n$$

6. Detta M la matrice di incidenza del grafo vale $\mathsf{L} = M^\intercal M$ (vedi sez. 2.3.1).

Dimostriamo la 5.:

Dim: Si vuole mostrare quindi che $x^\intercal \mathsf{L} x \geq 0, \forall x \in \mathbb{R}^n$. Si ha che

$$\begin{aligned} x^\intercal \mathsf{L} x &= \sum_{i=1}^{n}\sum_{j=1}^{n} \mathsf{L}_{i,j} x_i x_j = \\ &= \sum_{i=1}^{n} \Big(L_{i,i} x_i^2 + \sum_{j\neq 1} L_{i,j} x_i x_j \Big) = \\ &= \sum_{i=1}^{n} \Big(d_i x_i^2 + \sum_{j\in\mathcal{N}_i} -x_i x_j \Big) = \\ &= \sum_{i=1}^{n} \Big(\sum_{j\in\mathcal{N}_i} x_i^2 - x_i x_j \Big) \end{aligned}$$

si ha poi che la somma per tutti i nodi per ogni vicino può essere pensata circa come la somma su tutti gli archi

$$\sum_{i=1}^{n}\sum_{j\in\mathcal{N}_i} \approx \sum_{\{i,j\}\in\mathcal{E}}$$

in particolare, dato che queste sommatorie considerano tutti gli archi due volte (una volta come $\{i,j\}$ e una volta come $\{j,i\}$) si ha

$$\begin{aligned} x^\intercal \mathsf{L} x &= \sum_{i=1}^{n}\sum_{j\in\mathcal{N}_i} x_i^2 - x_i x_j \\ &= \sum_{\{i,j\}\in\mathcal{E}} (x_i^2 - x_i x_j + x_j^2 - x_i x_j) = \\ &= \sum_{\{i,j\}\in\mathcal{E}} (x_i - x_j)^2 \geq 0, \qquad \forall x. \end{aligned}$$

□

Seguendo l'esempio 5 avremo quindi, detto x il vettore complessivo degli stati $x = [x_1, x_2, x_3, x_4]^\intercal$:

$$\begin{aligned} x^\intercal \mathsf{L} x &= [x_1, x_2, x_3, x_4] \begin{bmatrix} -1 & -1 & 0 & 0 \\ -1 & 3 & -1 & -1 \\ 0 & -1 & 2 & -1 \\ 0 & -1 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \\ &= (x_1 - x_2)^2 + (x_2 - x_3)^2 + (x_3 - x_4)^2 + (x_2 - x_4)^2 \end{aligned}$$

2.3.1 Relazione tra Laplaciano e matrice d'incidenza

Supponendo di assegnare un valore x_i a ciascun nodo, allora $x^\intercal \mathsf{L} x$ **misura il disaccordo** (nel senso della distanza totale) tra nodi vicini. Nell'esempio se x_1 e x_2 fossero nello stesso stato ($x_1 = x_2$) si avrebbe distanza nulla.

Per calcolare $x^\intercal \mathsf{L} x$ si può costruire il vettore **vettore dei collegamenti** (seguendo l'esempio 1):

$$\xi = \begin{bmatrix} x_1 - x_2 \\ x_2 - x_3 \\ x_2 - x_4 \\ x_3 - x_4 \end{bmatrix} \tag{2.3.4}$$

e calcolarne il prodotto scalare (in norma 2):

$$\|\xi\|_2^2 = \xi^\intercal \xi = x^\intercal \mathsf{L} x \tag{2.3.5}$$

Si ha inoltre che, in generale, il vettore ξ puo essere ottenuto da x tramite la **matrice di incidenza** del grafo, M:

$$\xi = Mx \tag{2.3.6}$$

Definizione 6. *Matrice di incidenza:* La matrice d'incidenza M è una matrice $|\mathcal{E}| \times |\mathcal{N}|$ in cui ogni riga corrisponde a un arco e ogni colonna corrisponde a un nodo. Se la riga k di M è associata all'arco $\{i, j\}$ allora

$$M_{kl} \coloneqq \begin{cases} 1, & \text{per } l = i \\ -1, & \text{per } l = j \\ 0, & \text{altrimenti} \end{cases} \tag{2.3.7}$$

N.B: Ogni riga è definita a meno di un segno perche il grafo è non orientato.

Seguendo ancora l'esempio 5 si ha

$$\xi = \underbrace{\begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{bmatrix}}_{M} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{x} = \begin{bmatrix} x_1 - x_2 \\ x_2 - x_3 \\ x_2 - x_4 \\ x_3 - x_4 \end{bmatrix}$$

Si vede infine che:

$$x^\intercal \mathsf{L} x = \xi^\intercal \xi = (Mx)^\intercal (Mx) = x^\intercal M^\intercal M x \tag{2.3.8}$$

da cui abbiamo l'ultima proprietà del laplaciano, cioè che

$$\mathsf{L} = M^\intercal M \tag{2.3.9}$$

2.3.2 Cenni alla Teoria Spettrale

Poiche L è una matrice $n \times n$ simmetrica si ha che tutti i suoi autovalori $\lambda_1, \lambda_2, \ldots, \lambda_n$ sono reali.

A ciascun autovalore λ_i è associato un autovettore v_i con la proprietà

$$\mathsf{L}v_i = \lambda_i v_i \tag{2.3.10}$$

Una matrice simmetrica può sempre essere decomposta come

$$\mathsf{L} = V\Lambda V^\intercal \tag{2.3.11}$$

dove $\Lambda = diag\{\lambda_1, \dots, \lambda_n\}$ e $V = [v_1, \dots, v_n]$ è la matrice **ortogonale** degli autovettori (avente per colonne gli autovettori). Una conseguenza del fatto che V sia una matrice ortogonale è che $V^{-1} = V^\intercal$.
Quindi una matrice simmetrica è *completamente caratterizzata* dai suoi autovalori e dai suoi autovettori ed **il grafo quindi può essere codificato attraverso gli autovalori e gli autovettori del laplaciano**. Da qui si è sviluppata una teoria che ha come obiettivo lo studio degli autovalori e degli autovettori del laplaciano che prende il nome di Teoria Spettrale dei grafi.

Essendo $\{\lambda_1, \dots, \lambda_n\} \in \mathbb{R}$ possiamo ordinarli in modo crescente, supponendo, senza perdita di generalità, di avere questa struttura:

$$0 \leq \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n \tag{2.3.12}$$

dal momento che, essendo L semidefinita positiva, si ha che $\lambda_i \geq 0, \forall i \in \{1, \dots, n\}$.
In realtà **per ogni grafo vale**:

$$\lambda_1 = 0 \text{ con autovettore associato } v_1 = \frac{1}{\sqrt{n}}\begin{bmatrix}1\\1\\\vdots\\1\end{bmatrix} = \frac{1}{\sqrt{n}}\underline{1} \quad \text{per costruzione.} \tag{2.3.13}$$

Infatti è facile vedere che $\mathsf{L}\underline{1} = 0$ quindi non ci da particolare informazioni sul grafo dal momento che è **vero per tutti**, infatti:

$$\mathsf{L}\underline{1} = \begin{bmatrix}\mathsf{L}_{11} + \mathsf{L}_{12} + \dots \mathsf{L}_{1n}\\\vdots\\\mathsf{L}_{n1} + \mathsf{L}_{n2} + \dots \mathsf{L}_{nn}\end{bmatrix} = \begin{bmatrix}0\\\vdots\\0\end{bmatrix} \tag{2.3.14}$$

perchè tutte le righe di L sommano a 0. Si ha poi quindi che

$$\mathsf{L}\alpha\underline{1} = \alpha\mathsf{L}\underline{1} = 0, \quad \forall\alpha \in \mathbb{R} \tag{2.3.15}$$

Un altro modo per vederlo è analizzando la forma quadratica

$$x^\intercal\mathsf{L}x = \sum_{\{i,j\}\in\mathcal{E}} (x_i - x_j)^2 \tag{2.3.16}$$

prendendo $x = \alpha\underline{1} = \begin{bmatrix}\alpha\\\vdots\\\alpha\end{bmatrix}$ si ha che

$$x^\intercal\mathsf{L}x = \sum_{\{i,j\}\in\mathcal{E}} (\alpha - \alpha)^2 = 0 \tag{2.3.17}$$

in cui quindi si ha **disaccordo nullo**.
Si può dimostrare poi che per qualsiasi grafo non orientato vale che il massimo degli autovalori

$$\lambda_n \leq 2d_{max} \tag{2.3.18}$$

dove $d_{max} = \max_i d_i$ (ovvero il grado massimo tra tutti i nodi). Vale quindi in definitiva che:

$$0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n \leq 2d_{max} \tag{2.3.19}$$

2.4 Connettività algebrica

Concentraimoci adesso su un oggetto fondamentale: il secondo autovalore λ_2. Questo prende il nome di **connettività algebrica del grafo** (anche detto *autovalore di Fiedler*) e ci da informazione appunto sulla connettività del grafo.

Per prima cosa vale che, per un grafo indiretto:

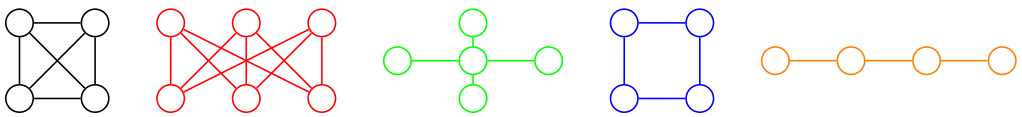
$$\lambda_2 > 0 \iff \text{il grafo è connesso} \tag{2.4.1}$$

Oltre a fornire un'informazione binaria (connesso-non connesso) ci da una **misura quantitativa** della connettività di un grafo, in quanto più il grafo è connesso più grande risulta λ_2. Questo autovalore tiene conto *sia* della velocità con cui si diffonde l'informazione *sia* della robustezza rispetto a disconnessioni di nodi/archi.
Vale infatti che

$$\lambda_2 \leq \mathcal{V}_n \leq \mathcal{V}_e \tag{2.4.2}$$

dove $\mathcal{V}_n, \mathcal{V}_e$ sono, rispettivamente, la *node connectivity* e la *edge connectivity*.

Per grafi con struttura "semplice" λ_2 si calcola analiticamente, ad esempio nel *grafo completo* si ha $\lambda_2 = n$ (molto connesso), nel *grafo bipartito* si ha $\lambda_2 = n/2$ mentre nel *grafo a stella* si ha $\lambda_2 = 1$ (molto poco connesso).
In un *ciclo* si ha, con conti un po piu lunghi, $\lambda_2 = 2[1 - \cos(2\pi/n)]$, infatti quando n cresce il grafo perde connettività (il cos $\to 1$ e $\lambda_2 \to 0$). Per un *cammino* si ha $\lambda_2 = 2[1 - cos(\pi/n)]$.



Per grafi con struttura più complicata si vanno ad utilizzare tecniche di analisi numerica per ricavare il valore di λ_2.

Vediamo ora una proprietà della connettività algebrica:

$$\begin{cases} \text{Grafo connesso} \iff \lambda_2 > 0 \iff \text{l'autovalore in 0 } (\lambda_1) \text{ ha molteplicità 1} \\ \text{Grafo non connesso} \iff \lambda_2 = 0 \end{cases} \tag{2.4.3}$$

Dim: Consideriamo la forma quadratica

$$x^\intercal \mathsf{L} x = \sum_{\{i,j\}} (x_i - x_j)^2$$

abbiamo visto che $x^\intercal \mathsf{L} x = 0$ quando $x = \alpha \underline{1}$. L'autovalore in 0 ha molteplicità 1 se e solo se non ci sono vettori diversi da $\alpha \underline{1}$ tali che $x^\intercal \mathsf{L} x = 0$ (o equivalentemente $Lx = 0$), ovvero se e solo se il mio laplaciano si annulla solo in una direzione (se ci fossero altri vettori avrei altre direzioni in cui si annulla il laplaciano).
Nel caso di **grafo connesso** quindi, considerando quella forma quadratica, si ha che

$$x^\intercal \mathsf{L} x = 0 \iff x_i - x_j = 0, \qquad \forall \{i,j\} \in \mathcal{E}$$

ovvero che tutti i nodi assumano lo stesso valore:

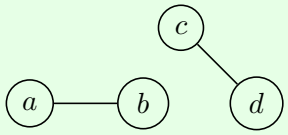
$$x^\intercal \mathsf{L} x = 0 \iff x_i = x_j$$

questo per ogni coppia di nodi, non necessariamente collegati da un arco. Per un grafo connesso si ha quindi che:

$$x^\intercal \mathsf{L} x = 0 \iff \exists \alpha : x_i = \alpha, \forall i$$

ovvero in cui l'autovalore in 0 ha molteplicità 1, che implica $\lambda_2 > 0$.
Nel caso di grafo **non connesso** invece $x^\intercal \mathsf{L} x$ si annulla anche quando non tutti i valori associati ai nodi sono coincidenti, questo implica che l'autovalore in 0 ha molteplicità > 1 e quindi che $\lambda_2 = 0$.

□



In questo semplice caso si ha

$$x^\intercal \mathsf{L} x = (a-b)^2 + (c-d)^2 = 0 \iff a = b \wedge c = d$$

e quindi non implica che tutti i valori debbano essere necessariamente uguali tra loro.

In generale vale che il numero n_0 di autovalori in 0 (autovalori nulli) di L è pari al numero di **componenti connesse** del grafo. Nell'esempio appena fatto si avrebbe $\lambda_1 = \lambda_2 = 0$ e $\lambda_3 > 0$.

In generale se si hanno N componenti connesse si può definire un laplaciano L_k per ogni componente connessa e (*eventualmente riordinando i nodi*) si ha che il laplaciano complessivo sarà una matrice diagonale a blocchi:

$$\mathsf{L} = \begin{bmatrix} \mathsf{L}_1 & 0 & \dots & 0 \\ 0 & \mathsf{L}_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mathsf{L}_N \end{bmatrix} \tag{2.4.4}$$

da cui si vede che gli autovalori di L sono gli autovalori di L_1, gli autovalori di $\mathsf{L}_2, \dots$, autovalori di L_N in cui si ha 1 autovalore in 0 per ogni componente connessa k (quindi si hanno N autovalori in

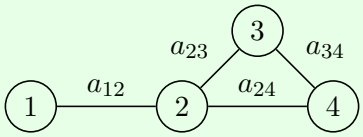
zero). Questo vuol dire che:

$$\operatorname{rank} \mathsf{L} = n - N \tag{2.4.5}$$

dove $n = |\mathcal{N}|$ è il numero di nodi e N il numero di componenti connesse.

Le considerazioni fatte fin ora valgono anche per **grafi pesati non orientati**: La definizione del laplaciano cambia solo minimamente.

Consideriamo ad esempio un semplice grafo pesato di questo tipo



Si ha una matrice di adiacenza

$$A = \begin{bmatrix} 0 & a_{12} & 0 & 0 \\ a_{12} & 0 & a_{23} & a_{24} \\ 0 & a_{23} & 0 & a_{34} \\ 0 & a_{24} & a_{34} & 0 \end{bmatrix}$$

e quindi un Laplaciano

$$\mathsf{L} = \begin{bmatrix} a_{12} & -a_{12} & 0 & 0 \\ -a_{12} & a_{12} + a_{23} + a_{24} & -a_{23} & -a_{24} \\ 0 & -a_{23} & a_{23} + a_{34} & -a_{34} \\ 0 & -a_{24} & -a_{34} & a_{24} + a_{34} \end{bmatrix}$$

In generale

$$\mathsf{L}_{ij} = \begin{cases} -a_{ij}, & j \in \mathcal{N}_i \\ \sum_k a_{ik}, & j = i, k \in \mathcal{N}_i \\ 0, & \text{altrimenti} \end{cases} \tag{2.4.6}$$

e continuano a valore le stesse proprietà, semplicemente entrano in gioco i pesi:

1. È simmetrico (nel caso indiretto).
2. Righe e colonne sommano a 0.
3. È semidefinito positivo.
4. $x^\intercal \mathsf{L} x = \sum_{\{i,j\} \in \mathcal{E}} a_{ij}(x_i - x_j)^2$.
5. $\lambda_1 = 0, \lambda_2 > 0 \iff$ il grafo è connesso.

2.5 Partizionamento di un grafo e clustering spettrale

Supponiamo di avere un grafo $\mathcal{G}$, vogliamo dividere il grafo in 2 parti: C_1, C_2 tali che $C_1 \cup \mathcal{C}_2 = \mathcal{N}$, $C_1 \cup C_2 = \emptyset$ e tali che $C_1 \neq \emptyset, C_2 \neq \emptyset$. Si vuole:

1. Avere pochi collegamenti tra nodi appartenenti a gruppi diversi.
2. Avere tanti collegamenti tra nodi appartenenti allo stesso gruppo.

Idea semplice: Si scelgono C_1, C_2 in modo da minimizzare il numero di collegamenti $\{i, j\}$ con $i \in C_1$ e $j \in C_2$. Questo *equivale a trovare il taglio del grafo che attraversa il minor numero di archi*, ovvero risolvere il problema del MinCut. Matematicamente quindi, data una partizione $\{C_1, C_2\}$ si può assegnare un valore x_i in questo modo:

$$x_i = \begin{cases} 1, & i \in C_1 \\ -1, & i \in C_2 \end{cases} \tag{2.5.1}$$

ovvero *assegnare un'etichetta* $\{+1, -1\}$ ad ogni nodo. Consideriamo ora il costo $J(x)$:

$$J(x) = \frac{1}{4} x^\intercal \mathsf{L} x = \frac{1}{4} \sum_{i,j} (x_i - x_j)^2 \tag{2.5.2}$$

da cui si ha che

$$(x_i - x_j)^2 = \begin{cases} 0, & \text{se } \{i, j\} \in C_1 \vee \{i, j\} \in C_2 \\ 4, & \text{se } i \in C_1, j \in C_2 \vee i \in C_2, j \in C_1 \end{cases} \tag{2.5.3}$$

ovvero che il costo J rappresenta il *numero di collegamenti tra nodi appartenenti a cluster diversi*, esprime cioè il **disaccordo totale**. Il problema del MinCut può essere quindi espresso in termini del Laplaciano come:

$$\begin{aligned} \min_{x \in \{-1,1\}^n} \quad & \frac{1}{4} x^\intercal \mathsf{L} x \\ \text{s.t} \quad & x \neq \underline{1} \wedge x \neq -\underline{1} \end{aligned}$$

dove il vincolo ci dice che i cluster *non devono essere vuoti*. Questo è un problema di ottimizzazione *combinatorio* perchè le variabili x_i sono discrete (in questo caso binarie), tuttavia può essere risolto in tempo polinomiale, attraverso l'algoritmo di Stoer-Wagner. In realtà questo algoritmo si usa principalmente per trovare la edge connectivity, non si usa nella pratica per trovare il MinCut dal momento che, per certi grafi, ha la tendenza a fare partizioni molto **sbilanciate**, in cui si ha che C_1 e C_2 possono avere dimensione molto diversa.

Idea 2: Vogliamo quindi imporre un certo bilanciamento. Questo può essere fatto ad esempio imponendo $|C_1| = |C_2|$, supponendo n pari, ovvero aggiungendo il vincolo:

$$\sum_{i=1}^{n} x_i = x^\intercal \underline{1} = 0 \tag{2.5.4}$$

Potremmo quindi pensare di risolver questo nuovo problema, in cui adesso è sufficiente solo questo vincolo:

$$\begin{aligned} \min_{x \in \{-1,1\}^n} \quad & \frac{1}{4} x^\intercal \mathsf{L} x \\ \text{s.t} \quad & x^\intercal \underline{1} = 0 \end{aligned}$$

Questo però porta a due *difficoltà*: fa esplodere la complessità computazione da polinomiale a *NP-hard* e, in secondo luogo, scegliere a priori la dimensione dei cluster non è consigliabile.

Idea 3: La soluzione può essere quella di rilassare il vincolo di integrità, svincolando da $x \in \{-1, 1\}^n$ a $x \in \mathbb{R}^n$ ma mantenendo il passaggio per $\{-1, 1\}^n$, ovvero preservando la norma imponendo $\|x\|^2 = x^\intercal x = n$. Per partizionare il grafo si risolve quindi infine il problema:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \frac{1}{4} x^\intercal \mathsf{L} x \\ \text{s.t} \quad & x^\intercal x = n \\ & x^\intercal \underline{1} = 0 \end{aligned}$$

Il vettore ottenuto sarà quindi formato da componenti continue: dopo aver risolto il problema si assegna quindi, ad esempio:

$$i \in C_1 \text{ se } x_i > 0, \qquad i \in C_2 \text{ se } x_i < 0 \tag{2.5.5}$$

come regola di decisione per assegnare il nodo i-esimo ad un cluster. La soglia non deve essere necessariamente a 0 ma può essere scelta secondo un qualche criterio (ad esempio seguendo tecniche di clustering).

Analizziamo la struttura della soluzione: Dato il Laplaciano L considero la connettività algebrica λ_2 e il corrispondente autoversore v_2:

$$\mathsf{L} v_2 = \lambda_2 v_2, \qquad v_2^\intercal v_2 = 1 \tag{2.5.6}$$

allora la **soluzione $\hat{x}$ del problema formulato** è data da:

$$\hat{x} = \frac{v_2}{\sqrt{n}} \tag{2.5.7}$$

ed il costo minimo $\hat{J}$ vale:

$$\hat{J} = \frac{\lambda_2}{n} \tag{2.5.8}$$

In generale, se si vuole suddividere il grafo in K gruppi, si può procedere in due modi:

1. Iterativamente: prima si divide in 2 gruppi, poi in 4, e così via. Per dividere un grafo in due sottografi si deve quindi

   - Calcolare L
   - Calcolare v_2
   - Esaminare, per ogni nodo i, la i-esima componente del vettore v_2 e decidere

   $$\begin{cases} i \in C_1 & \text{se } v_{2_i} > 0 \\ i \in C_2 & \text{se } v_{2_i} < 0 \end{cases} \tag{2.5.9}$$

2. Utilizzando *anche* gli autovettori successivi di L : $v_3, \ldots, v_K$. Partendo dalla decomposizione autovalore-autovettore:

   $$\mathsf{L} = V \Lambda V^\intercal$$

   si va a costruire, dalla matrice $n \times n$ ortonormale degli autovettori V, per ogni nodo i, un vettore

   $$x_i = \begin{bmatrix} v_{2_i} \\ v_{3_i} \\ \vdots \\ v_{K_i} \end{bmatrix}$$

e si partiziona il grafo usando una qualche tecnica di clustering (ad esempio k-means) per clusterizzare i vettori $x_1, \dots, x_N$.

Questo ci mostra come anche *gli autovalori del Laplaciano ci diano informazioni importanti sulla struttura topologica* del grafo.

2.5.1 Cenni al clustering spettrale

Queste idee si applicano anche al problema del clustering: Dati $z_1, z_2, \dots, z_N$ punti in $\mathbb{R}^n$, allora sulla base della distanza (o rispetto ad una funzione di similarità) si vuole dividerli in K gruppi (cluster) $C_1, \dots, C_K$ separati (tali cioè che $C_i \cap C_j = \emptyset$) in modo che:

- Punti appartenenti allo stesso cluster siano simili tra loro.
- Punti appartenenti a cluster diversi siano dissimili.

La somiglianza può essere misurata ad esempio attraverso la norma euclidea con $\|z_i - z_j\|$.

Un algoritmo tipico di clustering è il *k-means*, che si basa sull'idea di individuare K centroidi ed assegnare i punti al centroide più vicino. Questo algoritmo funziona quando i cluster sono effettivamente distribuiti intorno ai centroidi (come in figura 4), altrimenti no.
Tipicamente il k-means non funziona quando si devono separare cluster non convessi, come in figura 5 (in cui si ha una struttura di due cluster concentrici, che l'algoritmo non è in grado di individuare):

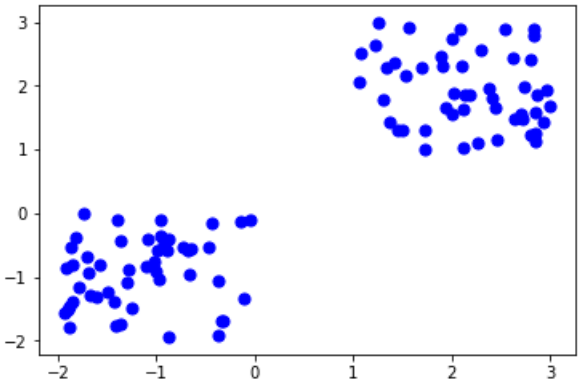


Figura 4: Cluster separati.

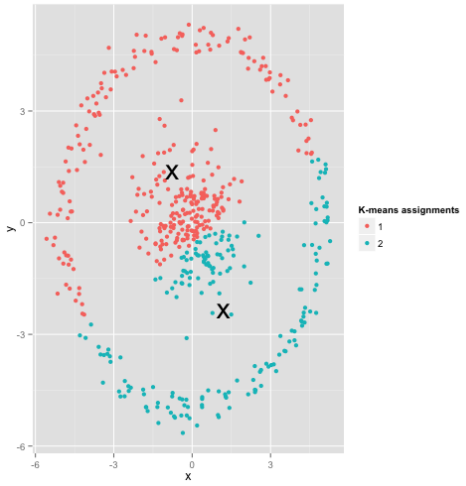


Figura 5: Cluster non convessi.

Il clustering spettrale invece risulta efficace:

1. Dati i punti $z_1, \dots, z_N$ costruisco un grafo $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ sulla base della loro similarità.

2. Si determina il Laplaciano L del grafo $\mathcal{G}$.

3. Si determina i primi K autovettori $v_1, v_2, \ldots, v_K$ (il primo non servirebbe dal momento che è sempre in zero).

4. Per ogni dato z_i a cui è associato il nodo i si costruisce il vettore delle componenti

$$x_i = \begin{bmatrix} v_{2_i} \\ v_{3_i} \\ \vdots \\ v_{K_i} \end{bmatrix} \tag{2.5.10}$$

5. Si applica un algoritmo di clustering (es. k-means) ai punti $x_1, \ldots, x_N$.

Per costruire $\mathcal{G}$ esistono diverse strategie:

1. Metodo ϵ-neighborhood: $\{i,j\} \in \mathcal{E} \iff \|x_i - x_j\| \leq \epsilon$ in cui ϵ è un iperparametro. In questo modo si costruisce un grafo geometrico non pesato.

2. Metodo h-nearest-neeighbors: per ogni punto z_i si cercano gli h punti più vicini: $\{i,j\} \iff j$ è uno dei primi h vicini di i o i è uno dei primi h vicini di j. In questo modo si costruisce simmetricamente un grafo non orientato e non pesato.

3. Costruire un grafo completamente connesso e pesato: all'arco $\{i,j\}$ si assegna un peso a_{ij} che dipende dalla distanza tra z_i e z_j, ad esempio prendendo

$$a_{ij} = \exp\left\{-\frac{\|z_i - z_j\|^2}{2\sigma^2}\right\} \tag{2.5.11}$$

   in cui scegliere σ è meno critico che scegliere ϵ: quest'ultimo infatti ci porta a una connettività binaria: connesso o non connesso, con σ riusciamo a mitigare questo fenomeno andando ad utilizzare un parametro continuo per indicare la forza della connessione. Questo può essere quindi visto come un rilassamento del primo metodo.

In alternativa esistono tecniche come *DBSCAN* che cercano di costruire un grafo già diviso in componenti separate tra loro anche se, come nel caso di DBSCAN, sono generalizzazioni del primo approccio e quindi si portano dietro la criticità della scelta di ϵ. In grafi costruiti con questo metodo per partizionare non serve calcolare $v_2, \ldots, v_K$ ma è sufficiente individuare le componenti connesse.

A volte invece di L si usa il Laplaciano normalizzato L_{nor} (anche se generalmente si hanno le stesse prestazioni della versione non normalizzata):

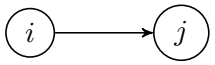
$$\mathsf{L}_{nor} \coloneqq D^{-1}\mathsf{L} = I - D^{-1}A \tag{2.5.12}$$

in cui ogni riga di L viene divisa per il grado d_i. Usare il Laplaciano normalizzato per partizionare il grafo rende il risultato meno dipendente dal grado.

2.6 Grafi orientati

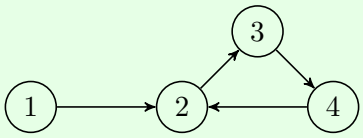
Abbiamo un grafo $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ con $\mathcal{N} = \{1, \dots, N\}$ ed $\mathcal{E}$ l'insieme delle coppie orientate $(i, j) \in \mathcal{N} \times \mathcal{N}$. Sia che:

- $(i, j) \in \mathcal{E} \iff$ c'è un arco che va da i a j.
- Il nodo i è un vicino di j ($i \in \mathcal{N}_j$) quando j riceve informazione da i (c'è un arco da i a j).



- Sia $d_i = |\mathcal{N}_i|$ il grado.
- La matrice di incidenza $D = diag\{d_1, d_2, \dots, d_N\}$ e la matrice di adiacenza A:

$$A = \begin{cases} 1, & \text{se } j \in \mathcal{N}_i \iff (j, i) \in \mathcal{E} \\ 0, & \text{altrimenti} \end{cases} \tag{2.6.1}$$



Si ha $\mathcal{N}_1 = \emptyset, d_1 = 0$, $\mathcal{N}_2 = \{1, 4\}, d_2 = 2, \mathcal{N}_3 = \{2\}, d_3 = 1$ e $\mathcal{N}_4 = \{3\}$ con $d_4 = 1$.

Per un grafo orientato esistono diversi concetti di connettività, in particolare un grafo orientato si dice:

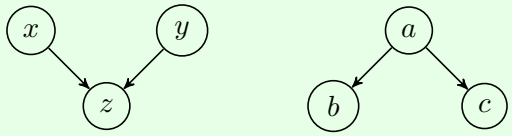
- **Debolmente connesso**: se la sua versione non orientata è connessa.
- **Fortemente connesso**: se $\forall i, j$ esiste un cammino orientato che va da i a j.

Il grafo dell'esempio è debolmente connesso ma non fortemente connesso. Ovviamente si ha che la connettività forte implica quella debole e non il viceversa. Nel nostro caso non saremo interessati ai debolmente connessi ma ci sta che la connettività forte sia una richiesta troppo restrittiva. Esistono quindi concetti di connettività intermedi, ad esempio:

- $\mathcal{G}$ si dice **quasi-fortemente connesso** se esiste almeno un nodo i da cui tutti gli altri nodi possono essere raggiunti. Nodi di questo tipo, se esistono, vengono detti nodi leader.

Il grafo dell'esempio quindi risulta quasi-fortemente connesso e il nodo 1 è il **nodo leader**.

Esempio 2.3. *Connettività quasi-forte e connettività debole*



Il secondo grafo è quasi-fortemente connesso (ma non fortemente connesso) mentre il primo no. I nodi x, y del primo prendono il nome di **nodi stubborn**.

Vale quindi **in definitiva**: *fortemente connesso* (tutti i nodi sono leader) $\implies$ *quasi-fortemente connesso* (esiste almeno un nodo leader) $\implies$ *debolmente connesso.*
Si ha poi una caratterizzazione equivalente di grafo quasi-fortemente connesso: *Un grafo orientato è quasi-fortemente connesso se e solo se contiene uno spanning tree.* Quindi avere un algoritmo per determinare uno spanning tree di un grafo permette di avere la caratterizzazione della connettività quasi forte.

Come cambia la definizione del Laplaciano per grafi orientati? Anche per i grafi orientati si definisce il Laplaciano L come

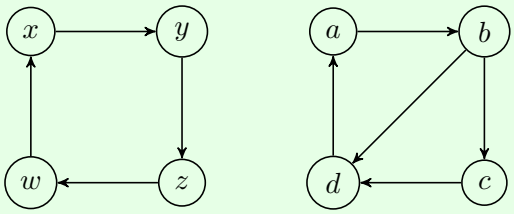
$$\mathsf{L} = D - A \tag{2.6.2}$$

Seguendo il grafo dell'esempio avremmo

$$\mathsf{L} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -1 & 2 & 0 & -1 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix} \tag{2.6.3}$$

Che **proprietà** abbiamo per L?

1. **Non è piu simmetrico** (è simmetrico solo per un grafo orientato).
2. Per costruzione *le righe sommano a zero.*
3. In generale le colonne non sommano a zero (non valendo piu la simmetria) ma, in particolare, le colonne sommano a zero se il grafo è bilanciato. (Un grafo si dice bilanciato quando per ogni nodo i abbiamo lo stesso numero di archi in ingresso e in uscita, cioè che tutti i nodi hanno la stessa importanza).



Il primo grafo è bilanciato mentre il secondo non lo è.

4. Poichè L non è simmetrico in generale ***non tutti gli autovalori sono reali.***

5. Poichè le righe sommano a zero allora vale

$$\mathsf{L}\underline{1} = 0$$

ovvero si ha sempre un autovalore $\lambda_1 = 0$ con autovettore

$$v_1 = \frac{1}{\sqrt{N}}\underline{1}$$

6. Per gli altri autovalori vale comunque:

$$0 \leq \Re\{\lambda_i\} \leq 2d_{max}$$

e si dimostra attraverso il Teorema di Gershgorin:

Idea della dimostrazione:
Data una qualunque matrice quadrata Q di dimensione $N \times N$ si ha che i suoi autovalori sono contenuti nell'unione $D_1 \cup D_2 \cup \dots D_N$ dove D_i rappresenta l'i-esimo disco di Gershgorin: è un cerchio con centro in Q_{ii} elemento diagonale e raggio dato dalla somma dei moduli degli elementi fuori diagonale

$$\sum_{j \neq i} |Q_{ij}| \tag{2.6.4}$$

Questo teorema è particolarmente utile quando la matrice che abbiamo è diagonale o è a diagonale dominante. Nel caso del Laplaciano si ha che l'elemento sulla diagonale è il grado del nodo i:

$$\mathsf{L}_{ii} = d_i \tag{2.6.5}$$

mentre gli elementi fuori diagonale sono dati da

$$\sum_{i \neq j} |\mathsf{L}_{ij}| = \sum_{j \in \mathcal{N}_i} |-1| = d_i \tag{2.6.6}$$

si ha quindi che il disco associato al nodo con grado massimo contiene tutti gli altri e, nel piano complesso, intersecherà i punti 0 e $2d_{max}$ (essendo un cerchio centrato in d_{max} e avente raggio d_{max})

7. Si dimostra che c'è solo un autovalore in zero se e solo se:

$$\lambda_i \neq 0, \forall i \neq 1 \iff \mathcal{G} \text{ è quasi fortemente connesso}$$

La cui dimostrazione completa si basa sul Teorema di Perron-Frobenius: l'idea è che per un grafo quasi-fortemente connesso (QFC) abbiamo $\mathsf{L}x = 0 \iff x = \alpha\underline{1}$, ovvero che $x_i = x_j, \forall i, j$. Per un grafo non QFC questo non è vero.

Tornando all'esempio 2.6 si ha che, nel primo grafo, il Laplaciano è dato da:

$$\mathsf{L}_1 = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

essendo una matrice trangolare inferiore si ha $\lambda_1 = 0, \lambda_2 = 1, \lambda_3 = 0$. Nel secondo grafo si ha

$$\mathsf{L}_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ -1 & -1 & 2 \end{bmatrix}$$

per cui si ha $\lambda_1 = \lambda_2 = 0$ e $\lambda_3 = 2$.

Per capire l'informazione che il Laplaciano dà si va ad assegnare dei valori ai nodi, ad esempio per il primo grafo

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

e, andando a vedere $\mathsf{L}x = 0$, si ha che

$$\mathsf{L}x = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = 0 \implies x_2 = x_1 \wedge x_3 = x_1$$

Il Laplaciano riflette internamente il modo in cui l'informazione fluisce nel grafo, del flusso di influenza tra nodi. In questo esempio abbiamo visto che per annullarsi x_2 e x_3 devono assumere lo stesso valore di x_1, che infatti è il **nodo leader**.

3 Sincronizzazione e Coordinamento nei Sistemi Multi-agente

Consideriamo un sistema multiagente cooperativo (SMAC) a struttura decentralizzata (SD) a tempo discreto:

$$x_i(t+1) = f_i(x_i(t), u_i(t)) \tag{3.0.1}$$

a tempo continuo sarebbe:

$$\dot{x_i}(t) = f_i(x_i(t), u_i(t)) \tag{3.0.2}$$

in cui $x(t) = [x_1(t), \ldots, x_N(t)]^\intercal$ è il vettore di stato collettivo. Ogni agente ha quindi a disposizione un vettore informativo a dimensione variabile $y_i(t), \forall t$:

$$y_i(t) = \begin{bmatrix} x_i(t) \\ x_j(t), j \in \mathcal{N}_i \end{bmatrix} \tag{3.0.3}$$

e, nella nostra trattazione, considereremo per ora obiettivi formulati in termini dell'insieme $\mathbb{S}$ delle configurazioni desiderate e dell'insieme $\mathbb{V}$ delle configurazioni ammissibili. Il nostro ruolo è quindi **progettare leggi di controllo** del tipo:

$$u_i(t) = \gamma_i(y_i(t)), \qquad i = 1, \ldots, N \tag{3.0.4}$$

dipendenti dal vettore informativo. Si dovrà avere che:

- $x(t) \in \mathbb{V}, \forall t$, supponendo che $x(0) \in \mathbb{V}$.
- $x(t) \to \mathbb{S}$ per $t \to \infty$.

Consideriamo due problemi introduttivi, il primo è quello della

3.1 Sincronizzazione

Non abbiamo vincoli, $\mathbb{V} = \emptyset$. L'insieme degli obiettivi è

$$\mathbb{S} = \{x : x_i = x_j, \forall i, j\} \tag{3.1.1}$$

ovvero si vuole che tutti gli agenti raggiungano lo stesso stato (uno stato comune). Ad esempio, nel caso di robot mobili se x_i rappresenta la posizione del robot il problema della sincronizzazione prende il nome di problema di **rendez-vous**: i robot, partendo da posizioni diverse, devono incontrarsi. Un caso particolare del problema della sincronizzazione è quello della sincronizzazione alla media. Anche qui non abbiamo vincoli e l'insieme degli obiettivi è dato da:

$$\mathbb{S} = \{x : x_i = \bar{x}\}, \qquad \bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i(0) \tag{3.1.2}$$

in cui gli stati degli agenti devono concergere alla media dei valori iniziali. Un'applicazione può essere quella del calcolo distribuito della media in una rete di centri di calcolo (es. sensori che eseguono misurazioni), in questo caso $x_i(t)$ rappresente il valore in memoria all'agente i al tempo t.

Per semplicità considereremo inizialmente agenti con dinamiche semplificate:

$$\begin{cases} \text{TC: } \dot{x_i}(t) = u_i(t) \\ \text{TD: } x_i(t+1) = u_i(t) \end{cases} \tag{3.1.3}$$

Nel caso TC si ha

$$x_i(t) = x_i(0) + \int_0^t u_i(r)dr \tag{3.1.4}$$

e per questo prende il nome di **dinamica a integratore**.

3.2 Consenso a tempo continuo

Siamo quindi nel contesto della dinamica a integratore

$$\dot{x_i}(t) = u_i(t), \qquad i = 1, \dots, N \tag{3.2.1}$$

L'obiettivo è la sincronizzazione, in cui ogni agente i conosce $x_i(t)$ e $x_j(t), \forall j \in \mathcal{N}_i$. L'idea semplice è che ogni agente confronti il suo stato $x_i(t)$ con quello dei vicini $x_j(t), j \in \mathcal{N}_i$ e agisca di conseguenza per modificare la propria condizione:

$$\begin{cases} \text{se } x_i(t) > x_j(t) \implies \text{l'agente } i \text{ deve diminuire il suo valore} \\ \text{se } x_i(t) < x_j(t) \implies \text{l'agente } i \text{ deve aumentare il suo valore} \\ \text{se } x_i(t) = x_j(t) \implies \text{l'agente } i \text{ deve mantenere il suo valore} \end{cases} \tag{3.2.2}$$

Essendo $\dot{x_i}(t) = u_i(t)$ si ha che $u_i(t)$ rappresenta la variazione dello stato per l'agente i. Nel caso di 1 vicino si ha che:

$$u_i(t) = x_j(t) - x_i(t) \tag{3.2.3}$$

Se quindi $x_i(t) > x_j(t)$ allora si ha $u_i(t) < 0$ e quindi l'agente i diminuisce il suo valore, se $x_i(t) = x_j(t)$ allora $u_i(t) = 0$ e quindi l'agente non modifica il suo valore. Se infine $x_i(t) < x_j(t)$ allora si ha $u_i(t) > 0$ e quindi l'agente i aumenta il suo valore. Nel caso di più vicini si ha (si fa un bilancio tra i valori):

$$u_i(t) = \sum_{j \in \mathcal{N}_i} x_j(t) - x_i(t) \tag{3.2.4}$$

e questa formula prende il nome di **legge di controllo del consenso**. Si ha quindi che la dinamica complessiva del SMA che dobbiamo studiare è data da:

$$\dot{x_i}(t) = \sum_{j \in \mathcal{N}_i} x_j(t) - x_i(t), \qquad i = 1, \dots, N \tag{3.2.5}$$

ed è il modo più semplice per risolvere un problema di sincronizzazione. In questa sommatoria il nodo i appare tante volte quante sono i vicini e posso quindi riscrivere la dinamica nella forma:

$$\begin{aligned} \dot{x_i}(t) &= -\sum_{j \in \mathcal{N}_i} x_i(t) + \sum_{j \in \mathcal{N}_i} x_j(t) = \\ &= -d_i x_i(t) + \sum_{j \in \mathcal{N}_i} x_j(t) = \\ &= -\Big[d_i x_i(t) - \sum_{j \in \mathcal{N}_i} x_j(t)\Big], \qquad i = 1, \dots, N \end{aligned}$$

in cui si vede emergere il Laplaciano: abbiamo infatti che

$$\begin{bmatrix} \dot{x_1}(t) \\ \vdots \\ \dot{x_N}(t) \end{bmatrix} = -\mathsf{L} \begin{bmatrix} x_1(t) \\ \vdots \\ x_N(t) \end{bmatrix} \tag{3.2.6}$$

e quindi che, in ultima istanza, la dinamica in termini dello stato collettivo sia del tipo:

$$\dot{x}(t) = -\mathsf{L}x(t) \tag{3.2.7}$$

Abbiamo quindi un sistema di questo tipo:

$$\begin{cases} \dot{x}_i(t) = u_i(t), & i = 1, \dots, N \\ u_i(t) = \sum_{j \in \mathcal{N}_i} [x_j(t) - x_i(t)] \end{cases} \tag{3.2.8}$$

Modelli di questo tipo sono anche rappresentativi per l'analisi della *dinamica delle opinioni*. In questo semplice modello si vede come ci sia la tendenza a convergere verso una stessa opinione (sincronizzazione), anche se ci sono modi piu raffinati per rappresentare questo comportamento.

Abbiamo visto che questo modello può essere scritto come

$$\dot{x}(t) = -\mathsf{L}x(t) \tag{3.2.9}$$

e adesso andremo ad analizzare come la connettività algebrica λ_2 influenzi la diffusione dell'informazione. Un paper di riferimento su quest'argomento è Olfati-Saber et al. 2007. **La velocità di convergenza** (con cui si diffonde l'informazione) è data da λ_2, vediamolo formalmente: Ricordiamo che, in uno stato di sincronizzazione (quando tutti gli stati hanno lo stesso valore)

$$x = \alpha \underline{1} \implies \mathsf{L}x = \mathsf{L}\alpha\underline{1} = 0 \tag{3.2.10}$$

si dice che lo stato sincronizzato è un *equilibrio* per il sistema:

$$\dot{x}(t) = -\mathsf{L}x(t) \text{ con } x(t) = \alpha\underline{1} \implies \dot{x}(t) = 0 \tag{3.2.11}$$

ovvero che lo stato complessivo è costante nel tempo.

Teorema 1. *Legge di controllo di consenso che garantisce la sincronizzazione alla media*: Sia G un grafo di comunicazione non orientato e connesso, allora la dinamica associata al consenso $\dot{x}(t) = -\mathsf{L}x(t)$ è tale che:

$$\lim_{t \to \infty} x_i(t) = \bar{x} \tag{3.2.12}$$

in cui $\bar{x}$ è la media dei valori iniziali:

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i(0), \qquad x(0) = [x_1(0), \dots, x_N(0)]^\intercal \tag{3.2.13}$$

Dim: $\dot{x}(t) = -\mathsf{L}x(t)$ è un sistema LTI, la soluzione è data da un'esponenziale di matrice:

$$x(t) = e^{-\mathsf{L}t}x(0) \tag{3.2.14}$$

Inoltre, per un grafo non orientato $\mathsf{L} = V\Lambda V^\intercal$ si ha $v_1 = \frac{1}{N}\underline{1}$ e $\lambda_1 = 0$. Sfruttando le proprietà dell'esponenziale di matrice:

$$e^{-\mathsf{L}t} = e^{-V\Lambda V^\intercal t} = -Ve^{-\Lambda t}V^\intercal \tag{3.2.15}$$

in cui

$$e^{-\Lambda t} = \begin{bmatrix} e^{-\lambda_1 t} & \dots & 0 \\ \vdots & & \vdots \\ 0 & \dots & e^{-\lambda_N t} \end{bmatrix} = diag\{e^{-\lambda_1 t}, \dots, e^{-\lambda_N t}\} \tag{3.2.16}$$

Quindi abbiamo che:

$$e^{-\mathsf{L}t}x(0) = Ve^{-\Lambda t}V^\intercal x(0) = [v_1, \dots, v_N] \begin{bmatrix} e^{-\lambda_1 t} & \dots & 0 \\ \vdots & & \vdots \\ 0 & \dots & e^{-\lambda_N t} \end{bmatrix} \begin{bmatrix} v_1 \\ \vdots \\ v_N \end{bmatrix} x(0) \tag{3.2.17}$$

Essendo $v_i v_j = 0, \forall i \neq j$ e $v_i v_i = 1, \forall i$ si ha che:

$$e^{-\mathsf{L}t}x(0) = \left(e^{-\lambda_1 t}v_1 v_1^\intercal + e^{-\lambda_2 t}v_2 v_2^\intercal + \dots + e^{-\lambda_N t}v_N v_N^\intercal\right)x(0) \tag{3.2.18}$$

Infine, passando al limite per $t \to \infty$ si ha, sfruttando il fatto che $\lambda_1 = 0$:

$$\lim_{t\to\infty} x(t) = \lim_{t\to\infty} e^{-\mathsf{L}t}x(0) = v_1 v_1^\intercal x(0) = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \frac{1}{\sqrt{N}}[1 \dots 1] \begin{bmatrix} x_1(0) \\ \vdots \\ x_N(0) \end{bmatrix} =$$

$$= \frac{1}{N} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \sum_{i=1}^{N} x_i(0) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} x_i(0)}_{\bar{x}} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} = \bar{x}$$

N.B: Per un grafo connesso

$$e^{-\lambda_2 t}, \dots, \lambda e^{-\lambda_N t} \tag{3.2.19}$$

sono tutte esponenziali di matrici convergenti a 0 (esponenzialmente). L'esponenziale che converge più lentamente è $e^{-\lambda_2 t}$ (se il grafo non fosse connesso non convergerebbe neanche, essendo 0) e quindi la velocità di convergenza dipende da λ_2: **maggiore λ_2 più veloce la convergenza**.

□

3.3 Consenso a tempo discreto

Siamo nel caso

$$x_i(t+1) = u_i(t), \qquad i = 1, \dots, N \tag{3.3.1}$$

Abbiamo quindi visto che nel caso a TC si possono sincronizzare gli agenti con una dinamica del tipo

$$\dot{x}_i(t) = \sum_{j\in\mathcal{N}_i} [x_j(t) - x_i(t)] \tag{3.3.2}$$

La prima idea può essere quella di discretizzare sostituendo la derivata con il rapporto incrementale (*discretizzazione di Eulero*):

$$\frac{d}{dt}x_i(t) = \dot{x}_i(t) \approx \frac{x_i(t+1) - x_i(t)}{\epsilon} \tag{3.3.3}$$

in cui $\epsilon > 0$ prende il nome di *passo di discretizzazione* (deve essere *sufficientemente piccolo*, lo vedremo dopo). Abbiamo quindi (seguendo il modello):

$$\frac{x_i(t+1) - x_i(t)}{\epsilon} = \sum_{j \in \mathcal{N}_i} [x_j(t) - x_i(t)] \tag{3.3.4}$$

da cui:

$$x_i(t+1) = x_i(t) + \epsilon \sum_{j \in \mathcal{N}_i} [x_j(t) - x_i(t)], \qquad i = 1, \dots, N \tag{3.3.5}$$

Questa uguaglianza può essere riscritta come

$$\begin{aligned} x_i(t+1) &= [x_i(t) - \epsilon \sum_{j \in \mathcal{N}_i} x_i(t)] + \sum_{j \in \mathcal{N}_i} \epsilon x_j(t) = \\ &= x_i(t)(1 - \epsilon \sum_{j \in \mathcal{N}_i} 1) \\ &= \underbrace{(1 - \epsilon d_i)}_{\pi_{ii}} x_i(t) + \sum_{j \in \mathcal{N}_i} \underbrace{\epsilon}_{\pi_{ij}} x_j(t) \end{aligned}$$

E quindi in generale il consenso a tempo discreto è esprimibile come:

$$x_i(t+1) = \pi_{ii} x_i(t) + \sum_{j \in \mathcal{N}_i} \pi_{ij} x_j(t) \tag{3.3.6}$$

ovvero come una media pesata tra il valore del nodo i e i valori dei vicini: la media pesata è in questo caso una combinazione convessa

$$\pi_{ii} + \sum_{j \in \mathcal{N}_i} \pi_{ij} = 1 \tag{3.3.7}$$

con $\pi_{ii} > 0, \pi_{ij} > 0, j \in \mathcal{N}_i$, per costruzione. Per quanto riguarda la scelta di ϵ si ha che, discretizzando il consenso a tempo continuo, i pesi sono dati da:

$$\begin{cases} \pi_{ii} = 1 - \epsilon d_i \\ \pi_{ij} = \epsilon \end{cases} \tag{3.3.8}$$

e quindi, imponendo i vincoli di positività, abbiamo che

$$\begin{cases} \pi_{ij} > 0 \implies \epsilon > 0 \\ \pi_{ii} > 0 \implies \epsilon < \frac{1}{d_i} \end{cases} \tag{3.3.9}$$

da cui si ha che la scelta di ϵ è vincolata all'interno di:

$$\epsilon \in \left(0, \frac{1}{d_{max}}\right) \tag{3.3.10}$$

Infine si ha che forma matriciale del consenso a TD è esprimibile con:

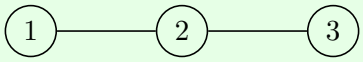
$$x(t+1) = \Pi x(t) \tag{3.3.11}$$

in cui Π prende il nome di **matrice di consenso** ed è costruita come:

$$\Pi_{ij} = \begin{cases} \pi_{ij}, & \text{se } j = i \vee j \in \mathcal{N}_i \\ 0, & \text{altrimenti} \end{cases} \tag{3.3.12}$$

Esempio 3.1. *Consenso a tempo discreto*

Consideriamo questo grafo



in cui si ha $d_1 = 1 = d_3, d_2 = 2$, quindi $\epsilon \in (0, 1/2)$:

$$\begin{cases} x_1(t+1) = (1-\epsilon)x_1(t) + \epsilon x_2(t) \\ x_2(t+1) = (1-2\epsilon)x_2(t) + \epsilon x_1(t) + \epsilon x_3(t) \\ x_3(t+1) = (1-\epsilon)x_3(t) + \epsilon x_2(t) \end{cases} \iff x(t+1) = \underbrace{\begin{bmatrix} 1-\epsilon & \epsilon & 0 \\ \epsilon & 1-2\epsilon & \epsilon \\ 0 & \epsilon & 1-\epsilon \end{bmatrix}}_{\Pi} x(t)$$

In generale quindi, ricapitolando, quando si discretizza il consenso tempo continuo con il metodo di Eulero si ha

$$\dot{x}(t) = -\mathsf{L}x(t) \implies \frac{x_i(t+1) - x_i(t)}{\epsilon} = \sum_{j \in \mathcal{N}_i} [x_j(t) - x_i(t)] \tag{3.3.13}$$

da cui, con $0 < \epsilon < 1/d_{max}$:

$$x(t+1) = x(t) - \epsilon \mathsf{L}x(t) = \underbrace{(I - \epsilon \mathsf{L})}_{\Pi} x(t) = \Pi x(t) \tag{3.3.14}$$

La matrice Π è una matrice stocastica, cioè è tale che:

- Tutti gli elementi sono non-negativi.
- Le righe per costruzione sommando a 1.

Se Π è simmetrica (*come in un grafo non orientato*) anche le colonne sommano a 1 e Π è una matrice doppiamente stocastica. Ricapitolando quindi abbiamo un:

Teorema 2. (*Dinamica del consenso a tempo discreto)*: Si ha

$$x(t+1) = \Pi x(t) \tag{3.3.15}$$

con Π matrice stocastica simmetrica:

$$\begin{cases} \pi_{ii} > 0 \\ \pi_{ij} > 0, j \in \mathcal{N}_i \\ \pi_{ii} + \sum_{j \in \mathcal{N}_i} \pi_{ij} = 1 \end{cases} \tag{3.3.16}$$

e, se il grafo di comunicazione è **non orientato e connesso** allora

$$\lim_{t\to\infty} x_i(t) = \bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i(0) \tag{3.3.17}$$

ovvero si ha, esponenzialmente, la sincronizzazione alla media (applicazione: calcolo distribuito della media).

Questo meccanismo presenta però una **difficoltà**: la scelta $\Pi = I - \epsilon \mathsf{L}$ va bene ma d_{max} è un'**informazione globale** e potremmo non conoscerla o potrebbe cambiare (ad esempio in un grafo tempo-variante). Esistono anche scelte alternative di ϵ che fanno uso esclusivamente di informazioni locali:

- *Pesi uniformi:*

$$\Pi_{ij} = \begin{cases} \frac{1}{d_i+1} \text{ se } j = i \vee j \in \mathcal{N}_i \\ 0 \text{ altrimenti} \end{cases} \tag{3.3.18}$$

  Seguendo l'esempio 3.3 si avrebbe:

$$\Pi = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/3 & 1/3 & 1/3 \\ 0 & 1/2 & 2/2 \end{bmatrix} \tag{3.3.19}$$

  Questo metodo **non va bene** perchè Π non è simmetrica e le colonne non sommano ad 1. Si converge, non alla media, bensì a una media pesata:

$$\lim_{t\to\infty} x_i(t) = \sum_{i=1}^{N} \frac{d_i}{\sum_{j=1}^{N} d_j} x_i(0) \tag{3.3.20}$$

  in cui il peso dell'agente i-esimo dipende dal grado d_i normalizzato sui vicini.

- *Pesi di Metropolis:* L'idea è quella di simmetrizzare i pesi uniformi attraverso uno scambio di informazione locale tra vicini. La matrice Π si costruisce nel seguente modo:

$$\Pi_{ij} = \begin{cases} \frac{1}{1+\max\{d_i,d_j\}}, & j \in \mathcal{N}_i \\ 1 - \sum_{k\in\mathcal{N}_i} \pi_{ik}, & j = i \text{ (si fa il complemento a 1)} \\ 0, & \text{altrimenti} \end{cases} \tag{3.3.21}$$

  Seguendo l'esempio 3.3 si avrebbe:

$$\Pi = \begin{bmatrix} 2/3 & 1/3 & 0 \\ 1/3 & 1/3 & 1/3 \\ 0 & 1/3 & 2/3 \end{bmatrix} \tag{3.3.22}$$

  Per un grafo non orientato Π è simmetrica e si può costruire semplicemente scambiando l'informazione sul grado con i vicini, il che è molto utile per applicazioni distribuite.

Abbiamo visto che Π può essere costruita a partire da L : $\Pi = I - \epsilon \mathsf{L}$ ma vale anche il viceversa: qualunque Π stocastica può essere interpretata in termini di un opportuno Laplaciano, infatti:

$$\Pi = I - \underbrace{(I - \Pi)}_{\mathsf{L}_\pi} = I - \mathsf{L}_\pi \tag{3.3.23}$$

in cui L_π è un Laplaciano pesato associato a Π. Sempre seguendo l'esempio 3.3 abbiamo che una generica matrice stocastica Π può essere scritta come

$$\Pi = \begin{bmatrix} \pi_{11} & \pi_{12} & 0 \\ \pi_{12} & \pi_{22} & \pi_{23} \\ 0 & \pi_{23} & \pi_{33} \end{bmatrix} \text{ con } \begin{cases} \pi_{11} + \pi_{12} = 1 \\ \pi_{12} + \pi_{22} + \pi_{23} = 1 \\ \pi_{23} + \pi_{33} = 1 \end{cases} \tag{3.3.24}$$

allora si può definire:

$$\begin{aligned} \mathsf{L}_\pi &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} \pi_{11} & \pi_{12} & 0 \\ \pi_{12} & \pi_{22} & \pi_{23} \\ 0 & \pi_{23} & \pi_{33} \end{bmatrix} = \\ &= \begin{bmatrix} 1-\pi_{11} & -\pi_{12} & 0 \\ -\pi_{12} & 1-\pi_{22} & -\pi_{23} \\ 0 & -\pi_{23} & 1-\pi_{33} \end{bmatrix} = \begin{bmatrix} \pi_{12} & -\pi_{12} & 0 \\ -\pi_{12} & \pi_{12}+\pi_{23} & -\pi_{23} \\ 0 & -\pi_{23} & \pi_{23} \end{bmatrix} \end{aligned}$$

3.4 Potenziali artificiali e coordinamento nei Sistemi Multi-Agente

Consideriamo una funzione $J(x)$, se ne voglio calcolare il minimo posso usare il metodo del gradiente: Partendo dall'inizializzazione $x(0)$ applico

$$x(t+1) = x(t) - \epsilon(t) \frac{\partial J}{\partial x}\Big|_{x=x(t)} \tag{3.4.1}$$

con $\epsilon(t)$ passo di discesa e t itereazione. Notiamo che, ponendo $\Pi = I - \epsilon\mathsf{L}$ si ha che **il consenso** $x(t+1) = \Pi x(t)$ (*con pesi che derivano dal Laplaciano*) è dato da:

$$x(t+1) = (I - \epsilon\mathsf{L})x(t) = x(t) - \epsilon\mathsf{L}x(t) \tag{3.4.2}$$

e **può essere visto come un metodo del gradiente per la funzione** $J(x) = \frac{1}{2}x^\intercal \mathsf{L}x$ con passo di discesa costante pari ad ϵ. Infatti

$$\frac{\partial J(x)}{\partial x} = \frac{\partial}{\partial x}\left\{\frac{1}{2}x^\intercal \mathsf{L}x\right\} = 2\frac{1}{2}\mathsf{L}x = \mathsf{L}x \tag{3.4.3}$$

e la funzione $J(x)$ viene detta **potenziale artificale**. Il sistema multi-agente evolve in modo da minimizzare questo potenziale e quindi il nostro obiettivo sarà progettare dei potenziali artificali in modo da ottenere il comportamento desiderato dal sistema. Questo dà un'interpretazione abbastanza chiara, dato che $J(x)$ può essere pensato **come il disaccordo complessivo tra gli agenti**:

$$J(x) = \frac{1}{2}x^\intercal \mathsf{L}x = \frac{1}{2}\sum_{\{i,j\}\in\mathcal{E}} (x_i - x_j)^2 \tag{3.4.4}$$

In generale per il consenso a tempo discreto:

$$\begin{aligned} x(t+1) = \Pi x(t) &= [I - (I - \Pi)]x(t) = \\ &= (I - \mathsf{L}_\pi)x(t) = x(t) - \mathsf{L}_\pi x(t) \end{aligned}$$

quindi definisco il potenziale artificiale come

$$J(x) = \frac{1}{2}x^\intercal \mathsf{L}_\pi x = \frac{1}{2}x^\intercal (I - \Pi)x = \frac{1}{2}\sum_{\{i,j\}\in\mathcal{E}} \pi_{ij}(x_i - x_j)^2 \tag{3.4.5}$$

ed ho un'equivalenza con il metodo del gradiente con peso costante $\epsilon(t) = 1$ e passo dato dal Laplaciano pesato. È importante sottolineare come **tutto sia definito mediante l'uso esclusivo dell'informazione locale**.

In generale considerando agenti del tipo:

$$x_i(t+1) = u_i(t), \qquad i = 1, \dots, N \tag{3.4.6}$$

e un generico problema di coordinamento

$$x(t) \to \mathbb{S} \text{ per } t \to \infty \wedge x(t) \in \mathbb{V}, \quad \forall t \tag{3.4.7}$$

con $\mathbb{S}$ insieme delle configurazioni desiderate e $\mathbb{V}$ quello delle configurazioni ammissibili vogliamo seguire quest'idea: *definire un potenziale artificiale $J(x)$ corrispondente ai miei obiettivi di controllo e quindi applicare una legge di controllo che minimizzi tale potenziale* (ad esempio il metodo del gradiente).

La domanda che sorge spontanea è: come si definisce $J(x)$? La proprietà che deve avere sono:

1. Deve essere sufficientemente smooth, ad esempio $J \in C^2$.
2. La legge di controllo deve essere distribuita, cioè $u_i(t)$ deve dipendere solo dallo stato locale $x_i(t)$ e dalla stato dei vicini $x_j(t), j \in \mathcal{N}_i$. Per fare in modo che sia così si definisce $J(x)$ come *somma di contributi*:

$$J(x) = \underbrace{\sum_{\{i,j\}\in\mathcal{E}} J_{\{i,j\}}(x_i, x_j)}_{\text{p.a. dell'arco } \{i,j\}} + \underbrace{\sum_{i=1}^{N} J_i(x_i)}_{\text{p.a. dell'agente } i} \tag{3.4.8}$$

   Si ha quindi

$$u_i(t) = x_i(t) - \epsilon(t)\left(\frac{\partial}{\partial x_i}\sum_{\{j,k\}\in\mathcal{E}} J_{\{j,k\}}(x_j, x_k) + \sum_{j=1}^{N} J_j(x_j)\right)\Bigg|_{x=x(t)} \tag{3.4.9}$$

   in cui *rimangono solo i termini in cui compare l'agente i*, cioè gli archi in cui l'agente i è una delle estremità. Si ha quindi, per $i = 1, \dots, N$:

$$u_i(t) = x_i(t) - \epsilon(t)\left(\sum_{j\in\mathcal{N}_i} \frac{\partial J_{\{i,j\}}}{\partial x_i}(x_i, x_j) + \frac{\partial}{\partial x_i} J_i(x_i)\right)\Bigg|_{x_i=x_i(t), x_j=x_j(t), j\in\mathcal{N}_i} \tag{3.4.10}$$

   Ad esempio, nel problema della sincronizzazione

$$J_{i,j}(x_i, x_j) = \pi_{ij}(x_i - x_j)^2 \tag{3.4.11}$$

   che ha un punto di minimo quando $x_i = x_j$

3. I minimi globali di $J(x)$ devono essere appartententi a $\mathbb{S}$:

$$\hat{\mathbb{X}} = \{\hat{x} \in \mathbb{X} : J(\hat{x}) \leq J(x), \forall x \in \mathcal{N}\} \tag{3.4.12}$$

allora deve essere:

$\hat{\mathbb{X}} \neq \emptyset$: devono esistere minimi globali

$\hat{\mathbb{X}} \subseteq \mathbb{S}$: tutti i minimi globali devono essere configurazioni desiderate

Poichè $J(x)$ deve essere una somma di termini (termini relativi ai collegamenti e termini relativi agli agenti)

$$J(x) = \sum_{\{i,j\} \in \mathcal{E}} J_{\{i,j\}}(x_i, x_j) + \sum_{i=1}^{N} J_i(x_i) \tag{3.4.13}$$

allora l'insieme $\mathbb{S}$ deve essere esprimibile in termini di condizioni su x_i, x_j con $\{i,j\} \in \mathcal{E}$. Nel caso della sincronizzazione abbiamo espresso $\mathbb{S}$ come

$$\mathbb{S} = \{x : x_i = x_j, \forall i, j\} \tag{3.4.14}$$

questa definizione è una definizione globale. Si ha però che per un grafo conneso può essere scritta il termini locali:

$$\mathbb{S} = \{x : x_i = x_j, \forall \{i,j\} \in \mathcal{E}\} \tag{3.4.15}$$

da cui si può ri-scrivere il p.a. per la sincronizzazione in termini locali:

$$J(x) = \sum_{\{i,j\} \in \mathcal{E}} \pi_{ij}(x_i - x_j)^2 \tag{3.4.16}$$

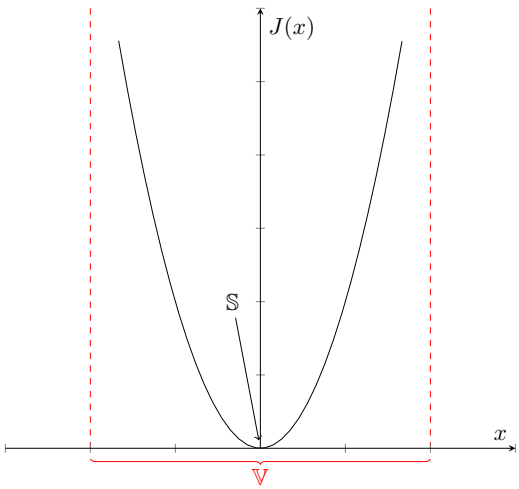
L'idea è: *si parte dall'insieme delle configurazioni desiderate, si guarda se questo è esprimibile in termini di informazione locale e sulla base di questo si definisce il potenziale artificiale.*

4. $J(x)$, se possbile, non deve avere minimi locali. Questo è garantito quando, ad esempio, $J(x)$ è una funzione convessa. È sufficiente garantire che i singoli termini $J_{\{i,j\}}$ e J_i siano convesse, dal momento che la somma di funzioni convesse è una funzione convessa.

5. Per garantire che $x(t) \in \mathbb{V}, \forall t$ allora $J(x)$ deve avere una *barriera di potenziale* che gli impedisca di uscire dall'insieme $\mathbb{V}$, ovvero

$$J(x) \to \infty \text{ per } x \to \partial\mathbb{V} \tag{3.4.17}$$

dove $\partial\mathbb{V}$ è la frontiera (contorno) dell'insieme degli stati ammissibili. La scrittura $x \to \partial\mathbb{V}$ vuol dire che la distanza tra x e $\partial\mathbb{V}$ tende a 0.



Facciamo alcune *osservazioni*:

- Non è sempre possibile evitare minimi locali anche se $J(x)$ la scegliamo noi, in particolare in presenza di ostacoli non convessi: in questi casi occorrono algoritmi per uscire dai minimi locali (ad esempio per aggirare l'ostacolo).
- La legge di controllo
$$x_i(t+1) = x_i(t) - \epsilon(t)\nabla_i J(x) \tag{3.4.18}$$
non tiene conto di eventuali vincoli su $u_i(t)$. In particolare si possono avere limitazioni su $x_i(t+1) - x_i(t)$, ad esempio nel caso i cui gli agenti siano agenti fisici (si potrebbe avere un ostacolo tra i due punti). Si introduce quindi una saturazione:
$$x_i(t+1) = x_i(t) - \frac{\tilde{x}_i(t)}{\|\tilde{x}_i(t)\|}\min\{\|\tilde{x}_i(t)\|, \tilde{x}_{max}\} \tag{3.4.19}$$
dove
$$\tilde{x}_i(t) = \epsilon(t)\frac{\partial J}{\partial x_i}\Big|_{x=x(t)} \tag{3.4.20}$$
e $\tilde{x}_{max}$ è la massima variazione ammessa per $x(t)$. Ad esempio per robot mobili:
$$\tilde{x}_{max} = V_{max} + T_s \tag{3.4.21}$$
dove V_{max} è la velocità massima e T_s è il tempo di campionamento.
- Nel caso generale, con dinamica del tipo
$$x_i(t+1) = f_i\Big(x_i(t), u_i(t)\Big) \tag{3.4.22}$$
si determina il valore $\hat{x}_i(t+1)$ desiderato mediante potenziale artificiale:
$$\hat{x}_i(t+1) = x_i(t) - \nabla_i J(x) \tag{3.4.23}$$
e poi si usa $\hat{x}_i(t+1)$ come riferimento per un controllo locale di basso livello.

TODO: Diagramma di controllo

3.5 Consenso per grafi orientati

Sia $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ un grafo orientato. Vogliamo connettere il problema del consenso a tempo continuo
$$\dot{x}_i(t) = \sum_{j\in\mathcal{N}_i} a_{ij}[x_j(t) - x_i(t)], \quad i = 1, \dots, N \tag{3.5.1}$$
con $a_{ij} > 0$ peso che indica l'influenza dell'agente i sull'agente j allo studio della dinamica di opinione su reti sociali.
Sia x_i l'opinione dell'agente i. Si ha
$$\begin{cases} x_i > 0 \text{ opinione positiva} \\ x_i = 0 \text{ opinione neutra} \\ x_i < 0 \text{ opinione negativa} \end{cases} \tag{3.5.2}$$

con a_{ij} che indica quanto pesa l'opinione j per i (può dipendere ad esempio dalla reputazione). La dinamica complessiva TC può essere quindi espressa come

$$\dot{x}(t) = -\mathsf{L}x(t) \tag{3.5.3}$$

con L laplaciano pesato (non necessariamente simmetrico). A tempo discreto si avrebbe, come in 3.3.15

$$x_I(t+1) = \pi_{ii}x_i(t) + \sum_{j\in\mathcal{N}_i} \pi_{ij}x_j(t) \tag{3.5.4}$$

con

$$\begin{cases} \pi_{ii} > 0 \\ \pi_{ii} + \sum_{j\in\mathcal{N}_i} \pi_{ij} = 1 \end{cases} \tag{3.5.5}$$

da cui, definendo come prima Π la matrice di consenso (in generale non simmetrica), seguendo 3.3.23 si ha

$$x(t+1) = \Pi x(t) = x(t) - \mathsf{L}_\pi x(t) \tag{3.5.6}$$

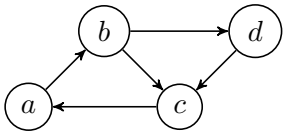
che, nel caso di un grafo non orientato (con L simmetrico) porta a una sincronizzazione alla media

$$\lim_{t\to\infty} x_i(t) = \bar{x} \tag{3.5.7}$$

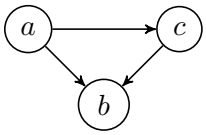
$$\bar{x} = \frac{1}{N}\sum i = 1^N x_i(0) \tag{3.5.8}$$

Cosa succede quando $\mathcal{G}$ è orientato? Distinguiamo 3 casi:

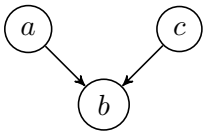
1. Grafo **fortemente connesso** (*tutti i nodi sono leader*): Ad esempio



2. Grafo **quasi fortemente connesso** (*esiste almeno un nodo leader*): Ad esempio



3. Grafo **debolmente connesso** (*non ci sono leader*): Ad esempio



Nel caso (1) il laplaciano L ha un solo autovalore in 0 (il comportamento è simile ad un grafo non orientato connesso). Si ha sincronizzazione con $\lim_{t\to\infty} x_i(t) = \bar{x}$ in cui tuttavia, in generale, $\bar{x}$ non coincide con la media dei valori iniziali $\frac{1}{N}\sum_{i=1}^N x_i(0)$ ma bensì con una media media pesata

$$\bar{x} = \sum_{i=1}^N \omega_i x_i(0) \tag{3.5.9}$$

con ω_i peso dell'agente i-esimo sul valore di sincronizzazione.
Ovviamente vale $\omega_i > 0,\ \forall i$ e $\sum_i \omega_i = 1$.
Il peso ω_i è una **misura dell'importanza** dell'agente i nel grafo, è cioè una misura di **centralità** dell'agente i-esimo.
Come si calcolano i pesi ω_i? Ricordando che L ha un autovalore $\lambda_1 = 0 \implies$ esiste autovettore destro $\mathsf{L}v_1 = \lambda_1 v_1 = 0$ con $v_1 = \frac{1}{N}\underline{1}$. Esiste poi un autovalore sinistro $w : w^\intercal \mathsf{L} = \lambda_1 w^\intercal$.
Il vettore di pesi w *si calcola quindi attraverso la soluzione di*

$$w^\intercal \mathsf{L} = 0 \tag{3.5.10}$$

Consideriamo la quantità:

$$w^\intercal x(t) = \begin{bmatrix} w_1 \dots w_N \end{bmatrix} \begin{bmatrix} x_1(t) \\ \vdots \\ x_N(t) \end{bmatrix} = \sum_{i=1}^{N} w_i x_i(t) \tag{3.5.11}$$

derivandola per ottenere la sua variazione si ha

$$\frac{d}{dt}[w^\intercal x(t)] = w^\intercal \frac{d}{dt} x(t) = w^\intercal \dot{x}(t) = -w^\intercal \mathsf{L} x(t) = 0 \tag{3.5.12}$$

Ovvero che la quantità $w^\intercal x(t)$ è costante. Si ha quindi che nell'evoluzione della dinamica

$$\dot{x}(t) = -\mathsf{L}x(t) \tag{3.5.13}$$

la quantità $w^\intercal x(t)$ si **conserva** e che ha quindi, per l'invarianza temporale, vale

$$w^\intercal x(0) = \lim_{t\to\infty} w^\intercal x(t) = w^\intercal \lim_{t\to\infty} x(t) \tag{3.5.14}$$

Inoltre, dal momento che si ha sincronizzazione

$$\lim_{t\to\infty} x(t) = \bar{x}\underline{1} = \begin{bmatrix} \bar{x}_1 \\ \vdots \\ \bar{x}_N \end{bmatrix} \tag{3.5.15}$$

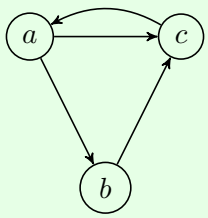
vale

$$\begin{aligned} & w^\intercal x(0) = w^\intercal \bar{x}\underline{1} = \bar{x} w^\intercal \underline{1} \implies \sum_{i=1}^{N} w_i x_i(0) = \bar{x} \sum_{i=1}^{N} w_i \\ & \implies \bar{x} = \frac{\sum_{i=1}^{N} w_i x_i(0)}{\sum_{i=1}^{N} w_i} = \sum_{i=1}^{N} \omega_i x_i(0) \\ & \implies \omega_i = \frac{w_i}{\sum_i w_i} \end{aligned} \tag{3.5.16}$$

Nel caso (2) c'è un solo autovalore in 0, ovvero si ha sincronizzazione

$$\lim_{t\to\infty} x_i(t) = \overline{x} \text{ con } \overline{x} = \sum^{N} \omega_i x_i(0) \text{ e } \omega_i = \begin{cases} > 0, & \text{se } i \text{ è leader} \\ = 0, & \text{se } i \text{ non è leader} \end{cases} \tag{3.5.17}$$

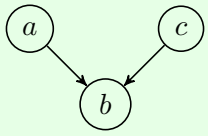
Consideriamo il caso seguente



Dove i leader sono a e b. Si ha $\omega_1 = \omega_2 = \frac{1}{2}$.

Nel caso (3) vi sono più autovalori in 0 il che implica che non ci sia sincronizzazione.

Consideriamo il seguente grafo



in cui

$$\begin{aligned} &x_1(t) = x_1(0) \\ &x_2(t) = x_2(0) \\ &x_3(t) \underset{t\to\infty}{\to} \frac{x_1(0)+x_2(0)}{2} \implies x_3(t) = w_1x_1(0) + w_2x_2(0) \end{aligned} \tag{3.5.18}$$

Si formano dunque **cluster di opinione**.

$$\begin{cases} \dot{x}_1(t) = 0 \\ \dot{x}_1 2(t) = 0 \\ \dot{x}_1(t) = x_1(t) + x_2(t) - x_3(t) \end{cases} \tag{3.5.19}$$

TODO: Disegnare il grafo con i cluster.
Tutti i nodi di C1 si sicronizzano con una media pesata dei valori iniziali dei nodi di C1. E così per C2, mentre quelli di C3 ad una media pesata sui valori di C1 e C2.

Tali modelli di dinamica di opinione sono i più semplici ma esistono anche modelli più raffinati, come ad esempio il **Bounded Confidence Model**, in cui $j \in \mathcal{N}_i$ influenza l'opinione di i se e solo se le opinioni di j e i non sono troppo distanti tra loro.

$$\dot{x}_i(t) = \sum_{j\in\mathcal{N}_i:|(x_i(t)-x_j(t)|<\rho} a_{ij}[x_j(t) - x_i(t)] \tag{3.5.20}$$

Con ρ intervallo di confidenza che esprime il distacco delle opinioni.
In modello di questo tipo si possono formare cluster di opinione anche in grafi fortemente connessi (avendo dipendenza dalle opinioni iniziali).

4 Sistemi Multi-Robot

Iniziamo con i sistemi multi-robot, in particolare cominceremo con il problema di controllo di formazione. Consideriamo N robot con dinamica a integratore

$$\dot{x}_i(t) = u_i(t), \qquad \forall i = 1, \ldots, N \tag{4.0.1}$$

in cui abbiamo che

- $x_i \in \mathbb{R}^n$ la posizione del robot i (con $n = 2$ o $n = 3$).
- u_i la velocità del robot i

Discretizzando l'equazione differenziale con T_s tempo di campionamento si ha

$$x_i(t+1) = x_i(x) + T_s u_i(t) \tag{4.0.2}$$

Gli obiettivi di controllo sono definiti attraverso un potenziale artificale $J(x)$ che viene minimizzato in modo da definire la legge di controllo

$$u_i(t) = -K_p \nabla_i J(x) \tag{4.0.3}$$

in cui K_p è un parametro detto *guadagno*.

A tempo continuo si ha quindi un *sistema a gradiente*

$$\dot{x}_i(t) = -K_p \nabla_i J(x), \qquad \forall i = 1, \ldots, N \tag{4.0.4}$$

che altro non è che la versione tempo discreto del metodo del gradiente. A tempo discreto si ha

$$x_i(t+1) = x_i(t) - T_s K_p \nabla_i J(x), \qquad \forall i = 1, \ldots, N \tag{4.0.5}$$

ovvero, praticamente, il metodo del gradiente per minimizzare $J(x)$ con passo di discesa dipendente dal tempo di campionamento e dal guadagno.

4.1 Controllo di Formazione

L'obiettivo è quello di portare i robot in una certa formazione desiderata specificata in termini di:

1. *Posizioni assolute:* in cui l'insieme delle configurazioni desiderate è dato da $\mathbb{S} = \{x : x_i = \xi_i, i = 1, \ldots, N\}$ dove ξ_i è la posizione desiderata per il robot i.
2. *Posizioni relative:* in cui l'insieme delle configurazioni desiderate è dato da $\mathbb{S} = \{x : x_i - x_j = \xi_{ij}, \forall \{i, j\} \in \mathcal{E}\}$ dove ξ_{ij} è la posizione relativa desiderata tra i robot i e j.
3. *Distanze relative:* in cui l'insieme delle configurazioni desiderate è dato da $\mathbb{S} = \{x : ||x_i - x_j|| = \delta_{ij}, \forall \{i, j\} \in \mathcal{E}\}$ dove δ_{ij} è la distanza desiderata tra i robot i e j.

- **Posizioni assolute**: Definiamo il potenziale come

$$J(x) = \sum_{i=1}^{N} J_i(x_i(t)) \tag{4.1.1}$$

dove

$$J_i(x) = \frac{1}{2}||x_i(t) - \xi_i||^2 \tag{4.1.2}$$

che ha un unico punto di minimo quando $x_i = \xi_i$. La legge di controllo è

$$u_i(t) = -K_p(x_i(t) - \xi_i) = K_p(\xi_i - x_i(t)) \tag{4.1.3}$$

Questo va bene in un contesto *centralizzato* in cui un decisore centrale dice a ciascun robot dove andare. Inoltre ogni robot deve operare nello stesso sistema di coordinate, dal momento fanno uso delle posizioni assolute. In questo caso non serve coordinamento per raggiungere la formazione ma il coordinamento è necessario per altri obiettivi (mantenimento della connessione, evitamento di collisioni,...) esplicitati attraverso l'aggiunta di termini al potenziale.

- **Posizioni relative**: In questo caso si scrive

$$J(x) = \sum_{\{ij\}\in\mathcal{E}} \frac{1}{2}||x_i(t) - x_j(t) - \xi_{ij}||^2 \tag{4.1.4}$$

che ha un unico punto di minimo quando $x_i(t) - x_j(t) = \xi_{ij}$. La legge di controllo è

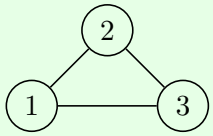
$$u_i(t) = -K_p \sum_{j\in N_i} (x_i(t) - x_j(t) - \xi_{ij}) = K_p \sum_{j\in N_i} (\xi_{ij} + x_j(t) - x_i(t)) \tag{4.1.5}$$

dove bisogna considerare, nella derivazione di $J(x)$, tutti i contributi dei vicini del nodo i-esimo. Per convenzione si ha poi $\xi_{ij} = -\xi_{ji}$. Anche in questo caso i robot devono operare in un sistema di coordinate comune. Occorre poi fare attenzione che la formazione sia ammissibile.

Definizione 7. *Formazione ammissibile:* Una formazione si dice ammissibile se e solo se esiste una realizzazione $x_i = x_i^*$ per $i = 1, \ldots, N$ che soddisfa tutti i vincoli $x_i^* - x_j^* = \xi_{ij}, \forall\{i, j\} \in \mathcal{E}$.

Esempio 4.1. *Formazione ammissibile in un triangolo*

In un grafo di questo tipo



si ha

$$\begin{cases} x_1 - x_2 = \xi_{12} \\ x_2 - x_3 = \xi_{23} \\ x_3 - x_1 = \xi_{31} \end{cases}$$

sommando le tre condizioni

$$(x_1 - x_2) + (x_2 - x_3) + (x_3 - x_1) = \xi_{12} + \xi_{23} + \xi_{31}$$

da cui si ha che la formazione è ammissibile se e solo se

$$0 = \xi_{12} + \xi_{23} + \xi_{31}$$

Condizioni analoghe valgono per tutti i cicli del grafo: *una formazione è ammissibile* **se e solo** se il sistema di equazioni lineari

$$x_i - x_j = \xi_{ij}, \quad \forall \{i,j\} \in \mathcal{E} \tag{4.1.6}$$

ammette soluzione. Questo può essere scritto in termini della matrice di incidenza M del grafo:

$$Mx = \xi \tag{4.1.7}$$

dove Mx è il vettore di tutte le differenze $x_i - x_j$ e ξ è il vettore delle differenze desiderate ξ_{ij}. In definitiva una formazione è ammissibile se e solo se

$$\text{rank}[M|\xi] = \text{rank}(M) \tag{4.1.8}$$

Se il grafo è connesso e la formazione è ammissibile allora tutte le realizzazioni $x_i = x_i^*, i = 1, \ldots, N$ che soddisfano i vincoli differiscono solo per una traslazione τ. Infatti si ha

$$x_i^* - x_j^* = \xi_{ij}, \forall \{i,j\} \in \mathcal{E} \tag{4.1.9}$$

se si effettua una traslazione $x_i^* \to x_i^* + \tau, \forall i$ si ha

$$x_i^* + \tau - (x_j^* + \tau) = x_i^* - x_j^* = \xi_{ij} \tag{4.1.10}$$

ovvero che la formazione viene specificata a meno di una traslazione arbitraria. Questo è un vantaggio qualora si voglia far muovere i robot in formazione di una traslazione τ uguale per tutti. Dovendo mantenere i vincoli si ha che una rotazione della formazione non è invece ammessa, e questo è uno dei motivi per cui analizziamo anche le distanze relative.

- **Distanze relative**: Anche qui il potenziale è definito tra coppie di nodi

$$J(x) = \sum_{\{i,j\} \in \mathcal{E}} J_{ij}(x_i, x_j) \tag{4.1.11}$$

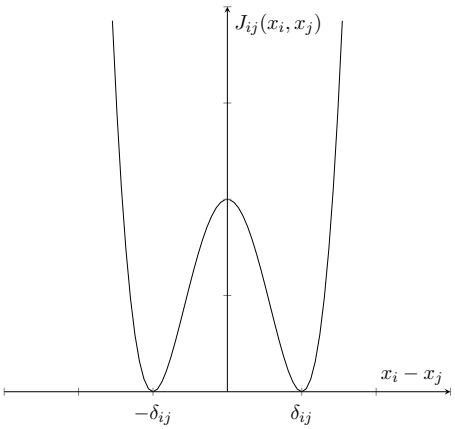
che si può specificare ad esempio come

$$J_{ij}(x_i, x_j) = \frac{1}{4}\Big(||x_i - x_j||^2 - \delta_{ij}^2\Big)^2 \tag{4.1.12}$$

che però è un oggetto non convesso. La legge di controllo quindi è data da

$$u_i(t) = -K_p \sum_{j \in N_i} \Big(||x_i - x_j||^2 - \delta_{ij}\Big)(x_i - x_j) \tag{4.1.13}$$

In questo caso non è necessario un sistema di coordinate globali ma ciascun robot può operare nel suo sistema di coordinate. Anche in questo caso occorre stare attenti che la formazione specificata sia ammissibile.



Riprendendo l'esempio 7 si ha che le le distanze desiderate

$$\begin{cases} ||x_1 - x_2|| = \delta_{12} \\ ||x_2 - x_3|| = \delta_{23} \\ ||x_1 - x_3|| = \delta_{13} \end{cases}$$

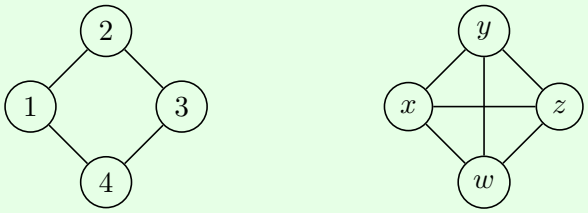
inducono che la formazione sia ammissibile se e solo se valgono le disuguaglianze triangolari

$$\begin{cases} \delta_{12} + \delta_{23} \leq \delta_{13} \\ \delta_{12} + \delta_{13} \leq \delta_{23} \\ \delta_{13} + \delta_{23} \leq \delta_{12} \end{cases}$$

In generale però, date le distanze δ_{ij} capire se la formazione è ammissibile non è banale come nell'esempio, dovendo risolvere un sistema di equazioni non-lineari. Normalmente si procede al contrario: prima si assegna una realizzazione $x_i = x_i^*$ e poi si calcolano le distanze (in questo modo la formazione è ammissibile per costruzione). Anche se la formazione è ammissibile e il grafo è connesso la formazione risultante può essere non ben definita.

Esempio 4.2. *Definizione di formazioni*

Consideriamo due grafi



Nel primo caso la formazione non è ben definita. Applicando una trasformazione non rigida (deformazione), ad esempio avvicinando i nodi 1 e 3 si otterrebbe un rombo. La formazione è quindi non rigida. Come si rende rigida? Specificando i vincoli sulle diagonali, come nel secondo grafo, in cui mantenendo le distanze si possono effettuare traslazioni e/o rotazioni.

4.1.1 Rigidezza

Specificare in termini di distanze relative permette quindi anche di ammettere rotazioni della formazione.

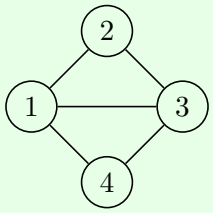
Definizione 8. *Formazione globalmente rigida:* Una formazione si dice **globalmente rigida** se, data una realizzazione $x_i = x_i^*, i = 1, \dots, N$ che soddisfa i vincoli

$$||x_i^* - x_j^*|| = \delta_{ij} \tag{4.1.14}$$

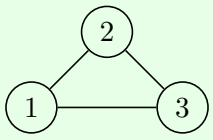
tutte le altre realizzazioni che soddisfano i vincoli differiscono solo per una trasformazione rigida (traslazione e/o rotazione). Saremmo interessati a formazioni globalmente rigida ma studiare questa proprietà è complicato, soprattutto in 3 dimensioni. Di solito si considera la condizione più debole di rigidezza locale.

Definizione 9. *Formazione localmente rigida:* Una formazione si dice **localmente rigida** se e solo se tutte le trasformazioni infinitesime che preservano le distanze sono rigide (ovvero non ammettono deformazioni). Questa proprietà al contrario della rigidezza globale può essere studiata in modo semplice.

Esempio 4.3. *Rigidezza di una formazione*



Questa formazione non è rigida in $\mathbb{R}^3$, dal momento che può essere piegata unendo i nodi 2 e 4. La rigidezza dipende quindi dallo spazio euclideo che si considera. Se si avessero dei robot volanti dovremmo tenerlo in considerazione. Inoltre questa formazione si dice *localmente rigida* in $\mathbb{R}^2$ dal momento che non si possono applicare deformazioni infinitesime (deformazioni con continuità) senza modificare le distanze. Se si sovrapponesse il nodo 2 con il nodo 4



si avrebbe che questa non è una *globalmente rigida* perchè esistono altre realizzazioni con formazioni diverse con le stesse distanze.

Fissata una realizzazione $x_i = x_i^*$ che soddisfa i vincoli, $\forall \{i, j\} \in \mathcal{E}$

$$||x_i^* - x_j^*|| = \delta_{ij} \tag{4.1.15}$$

si considerano informazioni infinitesime

$$\begin{cases} x_i = x_i^* + dx_i \\ x_j = x_j^* + dx_j \end{cases} \tag{4.1.16}$$

applicando una trasformazione quadratica sui vincoli (per la differenziabilità) e derivando entrambi i membri si ha

$$2(dx_i - dx_j)^\intercal (x_i^* - x_j^*) = 0 \tag{4.1.17}$$

quindi per mantenere la distanza costante tale variazione deve essere nulla. Le deformazioni infinitesime che preservano le distanze devono soddisfare i vincoli

$$(dx_i - dx_j)^\intercal (x_i^* - x_j^*) = 0, \qquad \forall \{i,j\} \in \mathcal{E} \tag{4.1.18}$$

Questo è un sistema di equazioni lineari che può essere scritto come, definendo il vettore delle deformazioni complessive $dx = [dx_1, \dots, dx_N]^\intercal$ e fissando la **matrice di rigidezza** $R(x^*)$, con $x^* = [x_1^*, \dots, x_N^*]^\intercal$ si ha

$$R(x^*)dx = 0 \tag{4.1.19}$$

La formazione è quindi localmente rigida nella realizzazione $x_1^*, \dots, x_N^*$ se e solo se tutte le soluzioni dx del sistema sono trasformazioni rigide (ovvero tutte le distanze sono preservate).
In $2D$ si hanno $2N$ (incognite) gradi di libertà mentre in $3D$ se ne hanno, ovviamente, $3N$. Dovendo ammettere solo trasformazioni rigide, ad esempio, in $2D$ devono rimanere 3 gradi di libertà. Sempre in $2D$ si ha che la formazione è localmente rigida se e solo se il sistema ha ∞^3 soluzioni. Questo è vero se e solo se

$$\text{rank } R(x^*) = 2N - 3 \tag{4.1.20}$$

dove 3 sono i gradi di libertà associati a una trasformazione rigida (un angolo e due traslazioni su x, y). In $3D$ si ha che la formazione è localmente rigida se e solo se si ha ∞^6 (avendo 6 gradi di libertà, x, y, z e gli angoli di Eulero) quindi se e solo se

$$\text{rank } R(x^*) = 3N - 6 \tag{4.1.21}$$

La rigidezza locale si studia quindi con una condizione di rango.

4.2 Mantenimento della connettività ed evitamento delle collisioni

4.2.1 Mantenimento della connettività

In situazioni reali la presenza di un collegamento $\{i,j\} \in \mathcal{E}$ tra due robot dipende dalle loro posizioni $x_i(t)$ e $x_j(t)$. Questo può essere dovuto a varie ragioni (raggio di comunicazione limitato, visibilità limitata, ...) e in pratica si ha un grafo tempo-variante $\mathcal{G} = (\mathcal{N}, \mathcal{E}(t))$. Studiamo il modello più semplice è dato da

$$\{i,j\} \in \mathcal{E}(t) \iff ||x_i(t) - x_j(t)|| \leq \rho \tag{4.2.1}$$

con ρ raggio di comunicazione. Il nostro obiettivo è: Supponendo $\mathcal{G}(0) = (\mathcal{N}, \mathcal{E}(0))$ connesso vogliamo $\mathcal{G}(t)$ connesso $\forall t$.
Idea semplice: se inizialmente è connesso l'idea è quella di continuare a preservare la connettività, considerando un sottoinsieme $\mathcal{E}_f \subseteq \mathcal{E}(0)$ di archi t.c. $\mathcal{G}_f = (\mathcal{N}, \mathcal{E}_f)$ sia connesso e si impone che

$$\mathcal{E}_f \subseteq \mathcal{E}(t), \qquad \forall t \tag{4.2.2}$$

Quindi si costruisce l'insieme dei vincoli (insieme di stati ammissibili) $\mathbb{V}_{con}$ come

$$\mathbb{V}_{con} = \{x : ||x_i - x_j|| \leq \rho, \quad \forall \{i,j\} \in \mathcal{E}_f\} \tag{4.2.3}$$

e l'obiettivo diventa $x(t) \in \mathbb{V}_{con}, \forall t$. Per soddisfare i vincoli costruiamo un potenziale artificiale a barriera:

$$J_{ij}^{con} = \begin{cases} 0, & \text{se } ||x_i - x_j|| \leq \rho_0 \\ K_{con}\left(\frac{1}{\rho - ||x_i - x_j||} - \frac{1}{\rho - \rho_0}\right)^{\beta}, & \text{se } ||x_i - x_j|| > \rho_0 \end{cases} \tag{4.2.4}$$

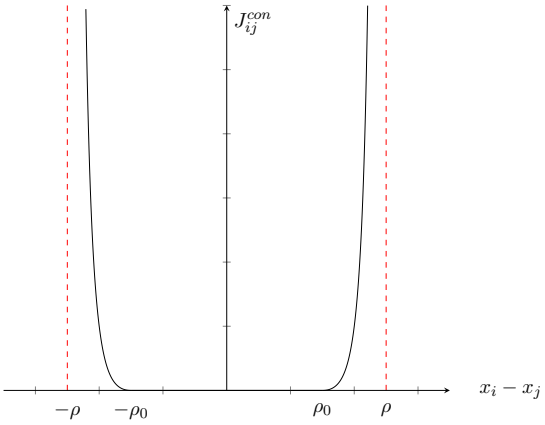
dove $\beta \in \mathbb{N}$ (tipicamente $\beta = 2, 3$). Il potenziale totale sarà quindi dato da

$$J(x) = J^{for}(x) + J^{con}(x) \tag{4.2.5}$$

dove il primo termine è il potenziale artificiale attrattivo per raggiungere la formazione desiderata e il secondo il potenziale artificiale repulsivo (barriera) per mantenere la connettività.

$$J^{con}(x) = \sum_{\{i,j\} \in \mathcal{E}_f} J_{ij}^{con}(x_i, x_j) \tag{4.2.6}$$

N.B: Questo potenziale artificiale è convesso perche $\mathbb{V}_{con}$ è un insieme convesso.



4.2.2 Evitamento delle collisioni

La distanza tra i robot deve essere sempre maggiore o uguale ad una distanza di sicurezza δ e quindi l'obiettivo sarà

$$||x_i - x_j|| \geq \delta, \quad \forall t, \quad \forall i \neq j \tag{4.2.7}$$

Questo obiettivo è relativo tra un robot e tutti gli altri (non come nel caso precedente che era limitato ai robot connessi). Questo definisce l'insieme di ammissibilità $\mathbb{V}^{col}$:

$$\mathbb{V}^{col} = \{x : ||x_i - x_j|| \geq \delta, \forall i \neq j\} \tag{4.2.8}$$

Il potenziale totale per evitare collisioni e mantenere le connessioni è definito come

$$J(x) = J^{for}(x) + J^{con}(x) + J^{col}(x) \tag{4.2.9}$$

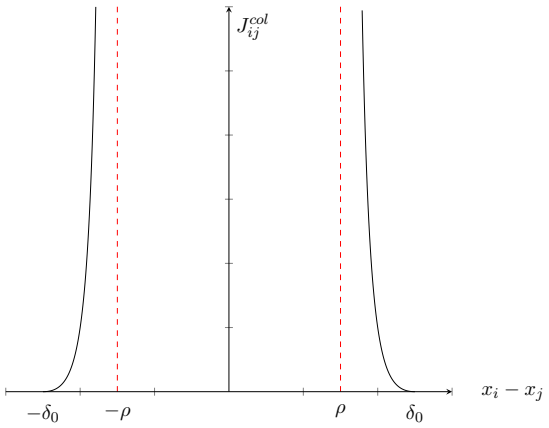
in cui J^{col} è un potenziale artificiale repulsivo per evitare collisioni:

$$J^{col}(x) = \sum_{i \neq j} J_{ij}^{col}(x_i, x_j) \tag{4.2.10}$$

e J_{ij}^{col} è il duale rispetto a quello precedente

$$J_{ij}^{col} = \begin{cases} 0, & \text{se } ||x_i - x_j|| \geq \delta_0 \\ K_{col}\left(\frac{1}{||x_i - x_j|| - \delta_0} - \frac{1}{\delta - \delta_0}\right)^{\beta}, & \text{se } ||x_i - x_j|| < \delta_0 \end{cases} \tag{4.2.11}$$

N.B: Questo potenziale artificiale è non convesso e quindi si hanno problemi legati ai minimi locali.



4.2.3 Evitamento delle collisioni con ostacoli

Anche in questo caso si definiscono potenziali artificiali repulsivi.
Caso semplice: Insieme finito di N_o ostacoli $\theta_1, \ldots, \theta_{N_o}$ per cui si possono calcolare in modo semplice la distanza tra un robot i e il k-esimo ostacolo:

$$d(x_i, \theta_k) = \min_{x \in \theta_k} ||x_i - x|| \tag{4.2.12}$$

Se, ad esempio, si rappresentano gli ostacoli con poligoni convessi si può definire un potenziale artificiale repulsivo per ogni ostacolo θ_k in funzione di $d(x_i, \theta_k)$ che sono della stessa forma del potenziale J_{ij}^{col} (dovendo evitare di andare a collidere) che interviene quando

$$d(x_i, \theta_k) \leq \hat{\delta} \tag{4.2.13}$$

Negli altri casi (ostacoli non convessi) si devono usare tecniche approssimate:

- Individuando i punti di potenziale collissioni all'interno di una circonferenza/sfera di raggio $\hat{\delta}$ (eventualmente discretizzando lo spazio di stato). Vedi Fig. 6.
- Definendo un potenziale artificiale repulsivo per ogni punto di collisione potenziale

4.2.4 Problema dei minimi locali

Siamo in presenza di potenziali artificiali non convessi nei casi di

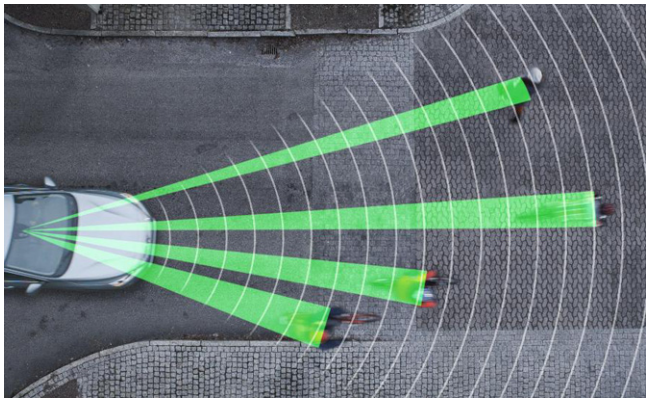


Figura 6: In questo caso i punti di collisione potenziale sarebbero dati dai ciclisti.

- Controllo di formazione basato sulle distanze.
- Evitamento delle collisioni.
- Evitamento delle collisioni con ostacoli.

possono esserci quindi minimi locali o comunque punti stazionari tali che

$$\nabla_i J(x) = 0 \tag{4.2.14}$$

ed in generale sono necessarie tecniche per risolvere il problema dei minimi locali, ad esempio:

1. Definire potenziali artificiali privi di minimi locali *(funzioni di navigazione, potenziali armonici, programmazione dinamica,...)*. Questo va bene in casi particolare dal momento che richiede la conoscenza dell'ambiente (ostacoli + altri robot). Queste soluzioni è quindi adatta nei casi di ambienti statici con approccio centralizzato e pianificazione *off-line*.
2. Evitare di finire nei minimi locali ad esempio fissando, mediante pianficazione di alto livello, degli obiettivi intermedi che evitano minimi locali. Anche questa tecnica è più adatta per pianificazione *off-line*. Un'altra tecnica è quella di utilizzare il gradiente stocastico:

   $$u_i(t) = -k_p \nabla_i J(x(t)) + \eta_i(t) \tag{4.2.15}$$

   dove $\eta_i(t)$ è una componente stocastica introdotta in modo da permettere una certo livello di esplorazione. In questo caso si ha un compromesso tra *exploitation* e *exploration* (come vedremo nel reinforcement learning).
3. Usare tecniche per fuggire dai minimi locali.

Queste due ultime tecniche richiedono un algoritmo di rivelazione, che ci dica quando si finisce in un minimo locale:

- **Tecniche di Wall-Following** (soprattutto in 2D per minimi locali dovuti alla presenza di ostacoli).

- **Random Walk** (soprattutto per minimi locali non dovuti a ostacoli). La componente stocastica interviene solo quando rivelo di essere in un minimo locale. Queste tecniche portano ad algoritmi corretti in probabilità (sono certo di arrivare al minimo globale, in tempo non garantito ovviamente).
- **Ostacoli virtuali**: si introduce un ostacolo virtuale in prossimità del minimo che introduce un potenziale repulsivo che fa sfuggire il robot (funziona in casi semplici).
- **Tecniche di pianificazione ad alto livello**: algoritmi di ricerca, backtracking, subgoals...

Nella pratica si ha una combinazione di queste tecniche. Ad esempio *wall following* combinato a *random walk* oppure tecniche di *reinforcement learning*.

4.3 Generalizzazione ad altri modelli di robot mobili

Abbiamo visto il modello a singolo integratore

$$\dot{x}_i(t) = u_i(t) \tag{4.3.1}$$

ma nella pratica non è sempre vero che si assegni direttamente la velocità (si pensi alla guida di una macchina: non si assegna la velocità, si preme l'acceleratore). La difficoltà di questo modello è quindi quella di supporre che si possa assegnare $\dot{x}_i(t)$ con il controllo. Un modello più realistico è il modello a **doppio integratore**:

$$\begin{cases} \dot{p}_i(t) = v_i(t) \\ \dot{v}_i(t) = u_i(t) \end{cases} \tag{4.3.2}$$

quindi

$$x_i(t) = \begin{bmatrix} p_i(t) \\ v_i(t) \end{bmatrix} \tag{4.3.3}$$

con p, v rispettivamente posizione e velocità e u accelereazione. Si hanno quindi obiettivi di controllo:

- Per la posizione (es. controllo di formazione).
- Per la veloctià.

Il nostro potenziale artificiale $J(x)$ sarà quindi costituito da diverse parti: definendo $p = [p_1, \dots, p_N]^\intercal$, $v = [v_1, \dots, v_N]^\intercal$

$$J(x) = K_p J^{for}(p) + K_v J^{vel}(v) + J^{rep} \tag{4.3.4}$$

dove J^{for} è il p.a. dipendente dalle posizioni per il controllo di formazione e J^{vel} è un p.a. dipendente dalle velocità per il controllo della velocità. J^{rep} è un termine che rappresenta i potenziali artificiali repulsivi aggiunti per altri obiettivi. Tipiche scelte per $J^{vel}(v)$:

- Per regolare a zero la velocità:

$$J^{vel}(v) = \sum_{i=1}^{N} ||v_i||^2 \tag{4.3.5}$$

- Per portare la velocità a un valore desiderato v^o:

$$J^{vel}(v) = \sum_{i=1}^{N} ||v_i - v^o||^2 \tag{4.3.6}$$

  ovvero allineare le velocità (problema del *flocking*).

- Per allineare le velocità senza specificare una velocità desiderata v^o, ovvero un problema di consenso sulla velocità:

$$J^{vel}(v) = \frac{1}{2} \sum_{\{i,j\} \in \mathcal{E}} ||v_i - v_j||^2 \tag{4.3.7}$$

  Asintoticamente la velocità di ciascun agente converge al baricentro delle velocità iniziali:

$$\frac{1}{N} \sum_{i=1}^{N} v_i(0) \tag{4.3.8}$$

- Per allineare la velocità a quella di un leader: ad esempio il robot 1 è il leader e sceglie in modo indipendente il suo controllo $u_1(t)$, i robot $= 2, \ldots, N$ effettuano poi un consenso sulla velocità.

Sia il modello singolo integratore che il doppio integratore sono modelli lineari. Nella realtà le dinamiche dei robot sono non linari e l'esempio più semplice è il **modello uniciclo**:

$$x_i(t) = \begin{bmatrix} p_i(t) \\ \theta_i(t) \end{bmatrix} \tag{4.3.9}$$

con $p_i \in \mathbb{R}^2$ e θ_i orientazione dell'i-esimo robot. Il controllo è dato da

$$u_i(t) = \begin{bmatrix} v_i(t) \\ w_i(t) \end{bmatrix} \tag{4.3.10}$$

con $v, w \in \mathbb{R}$ rispettivamente la velocità di avanzamento e velocità di sterzo. Nel modello dell'uniciclo si ha che, detto $p_i = [p_x^i, p_y^i]^\intercal$

$$\begin{cases} \dot{p}_x^i(t) = v_i(t) \cos\theta_i(t) \\ \dot{p}_y^i(t) = v_i(t) \sin\theta_i(t) \\ \dot{\theta}_i(t) = w_i(t) \end{cases} \tag{4.3.11}$$

La difficoltà è che non si può assegnare direttamente $\dot{p}_i = [\dot{p}_x^i(t), \dot{p}_y^i(t)]^\intercal$ perchè il vettore velocità **dipende dall'orientazione**. La strategia di controllo può essere data da questi step:

1. Si considera un modello semplificato a singolo integratore

$$\begin{cases} \dot{p}_x^i(t) = \tilde{u}_{x,i}(t) \\ \dot{p}_y^i(t) = \tilde{u}_{y,i}(t) \end{cases} \implies \dot{p}_i(t) = \tilde{u}_i(t) \tag{4.3.12}$$

2. Si applica il metodo APF per calcolare $\tilde{u}_i(t)$.

3. Si applica un controllo di basso livello per passare dal vettore velocità desiderato al controllo effettuato:

$$\tilde{u}_i(t) = \begin{bmatrix} \tilde{u}_{x,i}(t) \\ \tilde{u}_{y,i}(t) \end{bmatrix} \rightarrow u_i(t) = \begin{bmatrix} v_i(t) \\ w_i(t) \end{bmatrix} \tag{4.3.13}$$

L'idea è: invece di controllare il punto di coordinate (p_x^i, p_y^i) (che non si può controllare) si controlla un punto a distanza b:

$$\begin{cases} \tilde{p}_x^i(t) = p_x^i(t) + b\cos\theta_i(t) \\ \tilde{p}_y^i(t) = p_y^i(t) + b\sin\theta_i(t) \end{cases} \tag{4.3.14}$$

da cui

$$\begin{aligned} \frac{d}{dt}\tilde{p}_x^i(t) = \frac{d}{dt}p_x^i(t) + b\cos\theta_i(t) = \dot{p}_x^i(t) - b\sin\theta_i(t)\dot{\theta}_i(t) = \\ = v_i(t)\cos\theta_i(t) - bw_i(t)\sin\theta_i(t) \end{aligned}$$

e infine

$$\begin{cases} \frac{d}{dt}\tilde{p}_x^i(t) = v_i(t)\cos\theta_i(t) - bw_i(t)\sin\theta_i(t) \\ \frac{d}{dt}\tilde{p}_y^i(t) = v_i(t)\sin\theta_i(t) + bw_i(t)\cos\theta_i(t) \end{cases} \tag{4.3.15}$$

da cui, scegliendo $u_i(t) = [v_i(t), w_i(t)]^\intercal$ si può assegnare liberamente il vettore velocità:

$$\frac{d}{dt}\tilde{p}_i(t) = \tilde{u}_i(t) \tag{4.3.16}$$

Questa tecnica prende il nome di **feedback linearization**.

4.4 Cenni al coverage e all'esplorazione

Supponiamo di dotare ciascun robot di un sensore con un campo di vista $FoV_i(x_i)$ dipendente dalla posizione x_i. L'obiettivo del problema di **coverage** è quello di posizionare i robot in modo che l'unione dei campi di vista

$$\bigcup_{i=1}^{N} FoV_i(x_i) \tag{4.4.1}$$

copra tutta l'area di interesse, supponendo di avere un numero N di robot tali da coprire tutta l'area d'interesse. Se i robot hanno FoV più o meno uguali allora si devono disporre i robot in modo più o meno uniforme nell'area d'interesse. Questo può essere fatto in modo distribuito sfruttando il concetto di **partizione di Voronoi**: date le posizioni $x_1, \ldots, x_N$ dei robot si può associare a ogni robot i una cella di Voronoi C_i

in cui

$$C_i = \{x : ||x - x_i|| \leq ||x - x_j||, \forall j \neq i\} \tag{4.4.2}$$

Le celle $C_1, \ldots, C_N$ definiscono una partizione dell'area di interesse e si ha che la loro unione ci dà tutta l'area. Il calcolo delle partizioni di Voronoi può essere fatto in **modo completamente distribuito**, essendo necessario conoscere unicamente la propria posizione e le posizioni dei robot vicini (appartenenti a celle contigue).

dove, in generale, $C_i(t)$ dipende dal tempo se i robot si muovono e il baricentro $b_i(t)$ della cella $C_i(t$ si può calcolare in forma chiusa quando l'area di interesse è un poligono convesso (questo implica

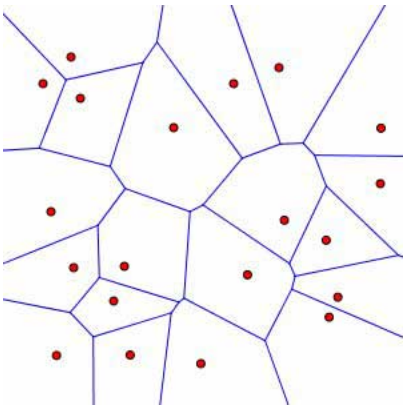
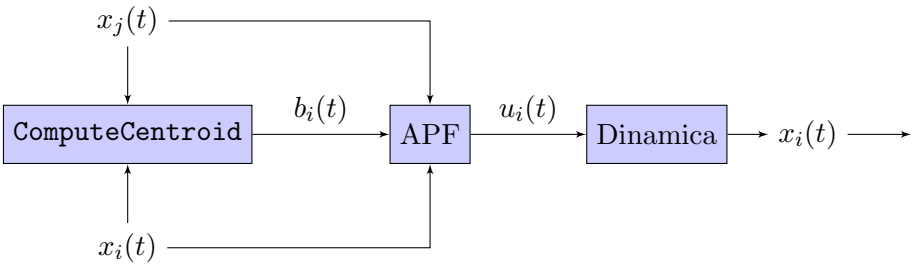


Figura 7: Esempio di tassellazione di Voronoi.

Algorithm 1 Coverage

Require: $x_i(t), x_j(t)$
Ensure: $j \in \mathcal{N}_i$
  for $t = 0, 1, \ldots$ **do**
    for all robot $i = 1, 2, \ldots, N$ **do**
      Compute $C_i(t)$
      Compute centroid $b_i(t)$
      Apply APF method with $J(x) = \sum_{i=1}^{N} ||x_i - b_i||^2 + J^{other}$
    end for
  end for

che anche le celle siano poligoni convessi) e noi consideriamo solo questo caso.
Il termine J^{other} rappresenta un certo numero di p.a. definiti per altri obiettivi. Si noti che $b_i(t)$ è una funzione di $C_i(t)$ e quindi di $x_1(t), \ldots, x_N(t)$.



Questo approccio si può applicare anche a *problemi di esplorazione*, in cui:

- Non è possibile coprire l'intera area di interesse con l'unione dei campi di vista.
- L'obiettivo è costruire/aggiornare una mappa dell'ambiente.

L'**idea** è: si costruisce la partizione di Voronoi e poi ciascun robot si muove all'interno della sua cella $C_i(t)$ verso aree inesplorate ad esempio in modo da massimizzare il guadagno di informazione

(definito opportunamente).
Servono quindi algoritmi per costruire/aggiornare in tempo reale la mappa dell'ambiente (campo di studio molto ampio, si fa con tecniche di inferenza Bayesiana). Inoltre sono necessari algoritmi per fondere in modo distribuito le mappe costruite dai singoli robot in modo da avere una mappa globale della regione esplorata (tecniche di fusione dell'informazione).

5 Fusione dell'Informazione nei Sistemi Multi-Agente

In molti contesti l'informazioni su variabili/vettori di interesse x è rappresentata da una pdf $p(x)$ (che puo essere pensata in uscita da un processo di stima/inferenza Bayesiana dai dati)

Tra le rappresentazioni tipiche per le $p(x)$ si ha:

- La *Gaussiana* (funziona nei casi piu semplici): ovvero si rappresenta l'incertezza in termini di media μ e matrice di covarianza Σ:

$$p(x) = \mathcal{N}(x; \mu, \Sigma) \propto e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)} \tag{5.0.1}$$

  in cui la media è il valore più probabile e la covarianza misura la confidenza sulla media.

- *Mistura di Gaussiane*:

$$p(x) = \sum_{c=1}^{n_c} \mathcal{N}(x; \mu_c, \Sigma_c) \tag{5.0.2}$$

  dove n_c è il numero di componenti gaussiane, σ_c la media della componente c-esima e Σ_c la sua covarianza. Non sempre è possibile rappresentare l'incertezza in termini di valore più probabile e dispersione ma si ha bisogno di un modello più flessibile e generale. Una mistura di gaussiane è un **approssimatore universale**, con cui si può approssimare con errore arbitrario un'arbitraria funzione $f(x)$ (continua su un compatto) al crescere di n_c, al variare di μ_c, Σ_c.

- *Insieme di particelle*: si ha un insieme di n_c particelle ciascuna con un suo peso w_c:

$$\{w_c, x_c\}_{c=1}^{n_c} \tag{5.0.3}$$

  dove x_c è la posizione della c-esima particella. Questa è una rappresentazione campionaria della pdf in cui il peso w_c rappresenta la probabilità di un valore x_c. Si ha

$$\sum_{c=1}^{n_c} w_c = 1 \tag{5.0.4}$$

  L'informazione è quindi codificata in termini di posizione e pesi delle particelle. Matematicamente si può vedere come somma di delte di Dirac

$$p(x) = \sum_{c=1}^{n_c} w_c \delta(x - x_c) \tag{5.0.5}$$

  La rappresentazione come insieme di particelle permette di svolgere più agevolmente il calcolo degli integrali (Metodo Monte Carlo):

$$\int_X p(x)dx = \int_X \sum_{c=1}^{n_c} w_c \delta(x - x_c)dx = \sum_{x_c \in X} w_c \tag{5.0.6}$$

e in generale

$$\mathbb{E}[f(x)] = \int f(x)p(x)dx = \int f(x)\sum_{c=1}^{n_c} w_c\delta(x-x_c)dx = \sum_{c=1}^{n_c} w_c f(x_c) \tag{5.0.7}$$

Quindi se si hanno N centri di elaborazione dati ognuno dei quali fornisce una certa pdf $p_i(x)$ con $i = 1, \ldots, N$, il nostro obiettivo è ora quello di combinare queste informazioni per ottenere una singola pdf fusa $\bar{p}(x)$ che riassuma *in modo ottimo* l'informazione contenuta nella pdf locali.
L'**idea** è: se abbiamo N stime/misure $x_1, x_2, \ldots, x_N$ per calcolare una stima fusa si può utilizzare la media campionaria

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{5.0.8}$$

oppure la mediana. Possiamo fare questo anche con le pdf e possiamo farlo in modo ottimo, ovvero definendo un criterio di ottimalità. Ricordando che la media campionaria

$$\bar{x} = \arg\min_x \sum_{i=1}^{N} \frac{1}{N}||x - x_i||^2 \tag{5.0.9}$$

e la mediana

$$\bar{x} = \arg\min_x \sum_{i=1}^{N} \frac{1}{N}||x - x_i||_1 \tag{5.0.10}$$

possono essere ottenute come il valore che minimizza la somma delle discrepanze (variazioni) rispetto ai valori da mediare $x_1, x_2, \ldots, x_N$, si considera una misura di discrepanza D tra pdf e si calcola $\bar{p}(x)$ come la densità che rende minima la discrepanza totale rispetto alle pdf locali.

$$\bar{p}(x) = \arg\min_p \sum_{i=1}^{N} \frac{1}{N} D(p, p_i) \tag{5.0.11}$$

La misura di discrepanza può essere presa come la divergenza di Kullback-Leibler (entropia relativa), anche se ci sono altre scelte

$$D_{KL}(p_1||p_2) = \int p_1(x)\log\frac{p_1(x)}{p_2(x)}dx \tag{5.0.12}$$

Si vede facilmente che

$$D_{KL}(p_1||p_2) = 0 \iff p_1(x) = p_2(x) \tag{5.0.13}$$

$$D_{KL}(p_1||p_2) \geq 0, \quad \forall p_1, p_2 \tag{5.0.14}$$

e in generale è tanto più piccola quanto p_1, p_2 sono simili. **N.B:** La divergenza KL non è una distanza perche non è simmetrica e non soddisfa la disuguaglianza triangolare.

Poichè non è simmetrica si può definire due concetti di media, rispetto alla entropia relativa:

- *Centroide sinistro* (cerca di massimizzare l'entropia):

$$\bar{p}_s(x) = \arg\min_p \sum_{i=1}^{N} \frac{1}{N} D_{KL}(p||p_i) \tag{5.0.15}$$

- *Centroide destro* (cerca di minimizzare la perdita di informazione):

$$\bar{p}_d(x) = \arg\min_p \sum_{i=1}^{N} \frac{1}{N} D_{KL}(p_i||p) \tag{5.0.16}$$

Si può anche estendere questi concetti alle medie pesate:

- Centroide sinistro:

$$\bar{p}_s(x) = \arg\min_p \sum_{i=1}^{N} \pi_i D_{KL}(p||p_i) \tag{5.0.17}$$

- Centroide destro:

$$\bar{p}_d(x) = \arg\min_p \sum_{i=1}^{N} \pi_i D_{KL}(p_i||p) \tag{5.0.18}$$

con $\pi_i > 0$ che sommano a 1. Quando tutti i pesi sono uguali e uniformi $\pi_i = 1/N$ si ritrova le definizioni precedenti. I due centroidi si possono calcolare in modo analitico:

$$\bar{p}_d(x) = \sum_{i=1}^{N} \pi_i p_i(x) \qquad (media \quad aritmetica) \tag{5.0.19}$$

e

$$\bar{p}_s(x) = \frac{\prod_{i=1}^{N} [p_i(x)]^{\pi_i}}{\int \prod_{i=1}^{N} [p_i(x)]^{\pi_i} dx} \qquad (media \quad geometrica \quad normalizzata) \tag{5.0.20}$$

N.B: I due centroidi hanno proprietà diverse: il centroide destro esegue una sorta di *or* tra pdf, tendendo a preservare i picchi mentre il centroide sinistro fa una sorta di *and* effettuando una fusione che privilegia i valori con probabilità non basse in tutte le pdf locali.

Dal punto di vista del calcolo:

- Il centroide destro si calcola facilmente per **tutte** le forme di pdf (Gaussiane, misture di gaussiane, insiemi a particelle, ...)

$$\bar{p}_d(x) = \sum_{i=1}^{N} \pi_i p_i(x) \tag{5.0.21}$$

  Ad esempio se tutte le pdf sono gaussiane $p_i(x) = \mathcal{N}(x; \mu_i, \sigma_i)$ si ha

$$\bar{p}_d(x) = \sum_{i=1}^{N} \pi_i \mathcal{N}(x; \mu_i, \sigma_i) \tag{5.0.22}$$

  (è abbastanza straightforward, non fa un gran riassunto dell'informazione e comporta un aumento di complessità.)

- Il centroide sinistro si calcola facilmente *solo in casi particolari* (es. Gaussiano) ma in generale (per $p_i(x)$ rappresentate da misture di gaussiane o insiemi di particelle) servono delle approssimazioni. Nel caso gaussiano, in cui le pdf sono $p_i(x) = \mathcal{N}(x; \mu_i, \Sigma_i)$ si ha

$$\bar{p}_s(x) = \mathcal{N}(x; \bar{\mu}, \bar{\Sigma}) \tag{5.0.23}$$

con

$$\bar{\Sigma}^{-1} = \pi_1 \Sigma_1^{-1} + \cdots + \pi_N \Sigma_N^{-1} \tag{5.0.24}$$

$$\mu = \bar{\Sigma}\left(\pi_1 \Sigma_1^{-1} \mu_1 + \cdots + \pi_N \Sigma_N^{-1} \mu_N\right) \tag{5.0.25}$$

La media fusa $\bar{\mu}$ è una media pesata delle medie $\mu_1, \ldots, \mu_N$ in cui i pesi sono dati da $\pi_i \Sigma_i^{-1}$. Un valore di Σ_i grande implica poca confidenza nel valor medio μ_i che genera una gaussiana più "spanciata". Ciascuna μ_i viene quindi pesata automaticamente in modo proporzionale alla confidenza Σ_i^{-1}.

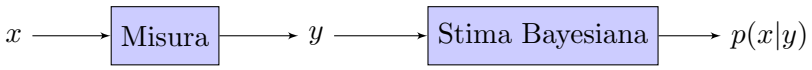
Il centroide sinistro è generalmente una fusione più accurata dell'informazione. **N.B:** In letteratura questa regola di fusione per gaussiane prende il nome di *covariance intersection*. Se definiamo per ogni gaussiana due quantità $\Omega_i = \Sigma_i^{-1}$ e $\omega_i = \Sigma_i^{-1}\mu_i$ che prendono il nome di matrice e vettore di informazione si ha che per la gaussiana fusa $\bar{p}_s(x)$ vale infine

$$\bar{\Omega} = \bar{\Sigma}^{-1} = \pi_1 \Omega_1 + \cdots + \pi_N \Omega_N \tag{5.0.26}$$

$$\bar{\omega} = \bar{\Sigma}^{-1}\bar{\mu} = \pi_1 \omega_1 + \cdots + \pi_N \omega_N \tag{5.0.27}$$

5.1 Elementi di Stima Bayesiana

Supponendo di essere interessati a stimare una quantità di interesse x si ha che, una volta ottenuto y a partire da un processo di misura/osservazione di x su un canale di misura, si vanno ad applicare delle tecniche di stima bayesiana (su y) al fine di ottenere la pdf condizionata $p(x|y)$.



In generale in questo contesto si ha quindi accesso ad una misurazione y e ad una pdf a priori $p(x)$, che riassume l'informazione su x disponibile prima dell'osservazione (si pensi ad esempio al monitoraggio di un'automobile: sapremo che la sua posizione sarà a terra e probabilmente nella carreggiata, e che la sua velocità sarà compresa in un certo range). Oltre a questi due elementi abbiamo anche un modello per il canale di misura, rappresentato dalla funzione di verosimiglianza $p(y|x)$.
Il nostro obiettivo è quello di calcolare $p(x|y)$: dal teorema di Bayes

$$p(x, y) = p(x|y)p(y) = p(y|x)p(x) \tag{5.1.1}$$

si ha

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} \tag{5.1.2}$$

da cui, essendo

$$p(y) = \int p(x,y)dx = \int p(y|x)p(x)dx \tag{5.1.3}$$

vale infine

$$p(x|y) = \frac{p(y|x)p(x)}{\int p(y|x)p(x)dx} \tag{5.1.4}$$

Questo è il caso **statico**, volendo fare inferenza nel tempo si deve estendere in questo modo: supponiamo di avere un sistema dinamico tempo discreto

$$x(t+1) = f\big(x(t), \xi(t), t\big) \tag{5.1.5}$$

dove $\xi(t)$ è il disturbo di processo al tempo t che modella l'incertezza nella transizione dallo stato $x(t)$ allo stato $x(t+1)$. Questa è una variabile aleatoria con pdf $p(\xi(t))$. Si suppone che $\{\xi_i(t)\}$ sia una **sequenza bianca**, ovvero una sequenza di disturbi $\xi(0), \xi(1), \dots$ è formata da tutte variabili aleatorie indipendenti. Si ha quindi $p(\xi_1(t), \dots, \xi_n(t) = p(\xi_1(t))p(\xi_2(t)) \dots, p(\xi_n(t))$. In questo modo stiamo assumendo che le realizzazioni passate del disturbo siano indipendenti dalle realizzazione future del processo (assunzione di Markov): data la pdf del disturbo $p_\xi(t)$ si può calcolare la **pdf di Markov**

$$\varphi(x(t+1)|x(t)) \tag{5.1.6}$$

ovvero la pdf che modella la probabilità di passare dallo stato $x(t)$ allo stato $x(t+1)$.
Oltre a questo abbiamo anche un'equazione di misura

$$y(t) = h\big(x(t), \eta(t), t\big) \tag{5.1.7}$$

dove $\eta(t)$ rappresenta l'errore di misura, che ci dà una rappresentazione matematica del canale di misura. Si suppone che anche $\{\eta_i(t)\}$ sia una sequenza bianca e che $\{\xi_i(t)\} \perp \{\eta_i(t)\}$ ovvero che le due incertezze siano indipendenti (incorrelate?) tra loro. Sotto queste ipotesi data la pdf del rumore $p(\eta(t))$ si può definire la **pdf di verosimiglianza**

$$g\big(y(t)|x(t)\big) \tag{5.1.8}$$

L'obiettivo della stima Bayesiana è quindi: date le osservazioni fino al tempo t, $y(0), y(1), \dots, y(t)$ calcolare la pdf a posteriori dello stato $x(t)$

$$p_{t|t}(x(t)) = p\big(x(t)|y(0), \dots, y(t)\big) \tag{5.1.9}$$

Nelle ipotesi fatte $p_{t|t}(x(t))$ si può calcolare ricorsivamente:

$$p_{t-1|t-1}(x(t-1)) = p\big(x(t-1)|y(0), \dots, y(t-1)\big)g\big(y(t)|y(t-1)\big) \tag{5.1.10}$$

eseguendo la predizione con la pdf di Markov $\varphi(x(t+1)|x(t))$ e ottenendo la pdf predettta

$$p_{t|t-1}(x(t)) = p(x(t)|y(0), \dots, y(t-1)) \tag{5.1.11}$$

Da qui, attraverso la verosimiglianza $g(y(t)|x(t))$ si ottiene $y(t)$ e con il teorema di Bayes si calcola la pdf di correzione

$$p_{t|t}(x(t)) = p(x(t)|y(0), \dots, y(t)) \tag{5.1.12}$$

6 Reinforcement Learning

Consideriamo un sistema dinamico stocastico che modella l'interazione tra agente e ambiente

$$x(t+1) = f\big(x(t), u(t), \xi(t)\big) \tag{6.0.1}$$

dove $\xi(t)$ è una variabile aleatoria che modella, come prima, l'incertezza sulla transizione tra $x(t)$ e $x(t+1)$. Invece $u(t)$ rappresenta il segnale di controllo/decisione/azione (deterministico) operato al tempo t. Facciamo le seguenti ipotesi (di *informazione completa*):

1. $\{\xi(t)\}$ è una sequenza bianca.
2. $p(\xi(t))$ è nota.
3. Lo stato $x(t)$ è accessibile/misurabile.

N.B: Sotto queste ipotesi si può definire la pdf di Markov di transizione dello stato (che ora dipende ovviamente anche da $u(t)$)

$$\varphi\big(x(t+1)|x(t), u(t)\big) \tag{6.0.2}$$

Questa ci dice la probabilità che lo stato al tempo $t+1$ sia $x(t+1)$ dato che lo stato al tempo t sia $x(t)$ a cui si applica l'azione $u(t)$.
Consideriamo ora un problema di controllo/decisione su un orizzonte (intervallo di tempo) finito T e si suppone di avere una funzione obiettivo da massimizzare

$$\mathcal{V} = \mathbb{E}_{\xi}\left[\sum_{t=0}^{T-1} r\Big(t, x(t), u(t))\Big) + r(T, x(t))\right] \tag{6.0.3}$$

in cui il primo termine prende il nome di **reward istantaneo** e il secondo quello di **reward finale** (che spesso è il più importante). Il valore atteso viene fatto sulla pdf $p(\xi(t))$. Il nostro obiettivo sarà determinare le tecniche per andare a risolvere questo tipo di problema.

6.1 Programmazione dinamica

Prima idea della programmazione dinamica: separare il passato dal futuro. Se abbiamo un problema di decisione su un orizzonte finito T, invece che considerare tutto l'orizzonte ci si restringe ad un ipotesi: detto $t \in [0, T]$ si suppone di essere nello stato $x(t)$ e, dal momento che il passato è passato, possiamo concentrarci solo sulle decisioni da prendere da t in poi, ovvero sul reward parziale da t in poi associato a una certa legge di controllo

$$u(\tau) = \gamma(x(\tau), \tau) \tag{6.1.1}$$

con $\tau \geq t$. Si riscrive quindi il reward come

$$\mathcal{V}^{\gamma}(x(t), t) = \mathbb{E}_{\xi(\tau), \tau \geq t}\Big\{\sum_{\tau=t}^{T-1} r(\tau, x(\tau), u(\tau)) + r(T, x(T))\Big\}_{u(\tau)=\gamma(x(\tau),\tau)} \tag{6.1.2}$$

e quindi come

$$\mathcal{V}^{\gamma}(x(t),t) = \mathbb{E}_{\xi(\tau),\tau\geq t}\Big\{ \sum_{\tau=t}^{T-1} r(\tau, x(\tau), \gamma(x(\tau),\tau)) + r(T, x(T))\Big\} \tag{6.1.3}$$

N.B: Possiamo separare passato e futuro perche abbiamo supposto la sequenza di disturbo $\{\xi(t)\}$ sia una *sequenza bianca*, il che implica che i disturbi futuri $\xi(t), \xi(t+1), \dots$ siano indipendenti dai passati $\xi(0), \xi(1), \dots, \xi(t-1)$. La cosa importante è che **tutto il passato è riassunto in** $x(t)$, ovvero nello stato/configurazione in cui ci si trova. A questo punto si definisce il reward parziale ottimo associato alla policy ottima

$$u(t) = \gamma^o(x(t), t) \tag{6.1.4}$$

e il reward complessivo (massimizzato prendendo ad ogni istante la decisione ottima)

$$\mathcal{V}^o(x(t),t) = \mathcal{V}^{\gamma^o}(x(t),t) = \tag{6.1.5}$$

$$= \max_{u(t)} \mathbb{E}_{\xi(t)} \max_{u(t+1)} \mathbb{E}_{\xi(t+1)} \dots \max_{u(T-1)} \mathbb{E}_{\xi(T-1)}\Big\{ \sum_{\tau=t}^{T-1} r(\tau, x(\tau), \gamma(x(\tau),\tau)) + r(T, x(T))\Big\} \tag{6.1.6}$$

N.B: L'alternanza tra max e $\mathbb{E}$ riflette la temporizzazione delle decisioni: al tempo t non si prendono tutte le decisioni fino a T ma semplicemente si prende la decisione migliore per arrivare al prossimo step (e le prossime decisioni verranno prese nel futuro).

La seconda idea è separare il presente e futuro: al tempo t si deve decidere solo $u(t)$:

$$\mathcal{V}^o(x(t),t) = \max_{u(t)} \mathbb{E}_{\xi(t)} \dots \mathbb{E}_{\xi(T-1)}\Big\{ r(t, x(t), u(t)) + \sum_{\tau=t+1}^{T-1} r(\tau, x(\tau), u(\tau)) + r(T, x(T))\Big\} = \tag{6.1.7}$$

$$= \max_{u(t)} \Big\{ r(t, x(t), u(t)) + \mathbb{E}_{\xi(t)} \max_{u(t+1)} \dots \Big[\sum_{\tau=t+1}^{T-1} r(\tau, x(\tau), u(\tau)) + r(T, x(t))\Big]\Big\} = \tag{6.1.8}$$

$$= \max_{u(t)} \Big\{ r(t, x(t), u(t)) + \mathbb{E}_{\xi(t)} \mathcal{V}^o(x(t+1), t+1)\Big\} \tag{6.1.9}$$

che prende il nome di **Equazione di Bellman** della programmazione dinamica. È un'equazione ricorsiva che dice che il reward ottimo parziale da t in poi si ottiene massimizzando il reward istantaneo sommato al valore atteso del reward ottimo futuro. La massimizzazione ci permette quindi di trovare il valore del reward ottimo ma anche ovviamente di identificare la decisione ottima

$$\gamma^o(x(t),t) = \arg\max_{u(t)} \Big\{ r(t, x(t), u(t)) + \mathbb{E}_{\xi(t)} \mathcal{V}^o(x(t+1), t+1)\Big\} \tag{6.1.10}$$

N.B: Si considera il reward ottimo da $t+1$ in poi mediando rispetto alla transizione incerta da t a $t+1$. Per calcolare $\mathcal{V}^o(x(t),T), \gamma^o(x(t),t)$ si deve conoscere quindi $\mathcal{V}^o(x(t+1),t+1), \forall x(t+1)$. È possibile? Nel contesto in cui ci siamo messi sì: essendo quella di Bellman un'equazione ricorsiva possiamo calcolare i $\mathcal{V}^o(x(t),t)$ procedendo ricorsivamente **indietro nel tempo**.
La programmazione dinamica si divide quindi in due fasi:

1. **Fase backward** di sintesi della policy (da svolgersi off-line):

$$\mathcal{V}^o(x(t),t) = \max_{u(t)} \Big\{ r(t, x(t), u(t)) + \mathbb{E}_{\xi(t)} \mathcal{V}^o(x(t+1), t+1)\Big\} \tag{6.1.11}$$

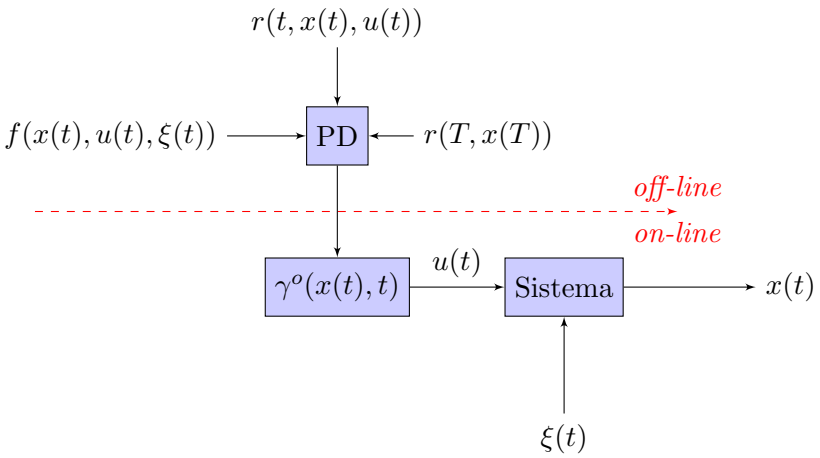
$$\gamma^o(x(t),t) = \arg\max_{u(t)} \Big\{ r(t, x(t), u(t)) + \mathbb{E}_{\xi(t)} \mathcal{V}^o(x(t+1), t+1)\Big\} \tag{6.1.12}$$

per $t = T-1, T-2, \ldots, 0$ con l'inizializzazione $\mathcal{V}^o(x(T), T) = r(x(T), T)$ al reward finale (che si ottiene alla fine, quando si arriva nello stato $x(T)$).

2. **Fase forward** di applicazione della policy (da svolgersi on-line):

$$u(t) = \gamma^o(x(t), t) \tag{6.1.13}$$

per $t = 0, 1, \ldots, T-1$ (via via che si giunge negli stati).

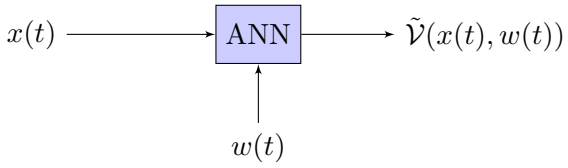


N.B: Nella fase backward si devono calcolare $\mathcal{V}^o, \gamma^o$ per ogni possibile stato $x(t) \in \mathbb{X}$ (spazio degli stati). È questa la difficoltà enorme della progrmmazione dinamica! Bisogna distinguere quindi alcuni casi applicabili:

- $\mathbb{X}$ discreto di cardinalità bassa/moderata (compatibile con le capacità di calcolo disponibile): Si può applicare la programmazione dinamica in modo esatto memorizzando $\forall t, \forall x(t)$ i valori $\mathcal{V}^o(x(t), t), \gamma^o(x(t), t)$.
- $\mathbb{X}$ insieme continuo in cui però si riesce a calcolare $\mathcal{V}^o(x(t), t)$ e $\gamma^o(x(t), t)$ in modo esatto (es. controllo ottimo lineare quadratico)
- $\mathbb{X}$ insieme continuo che può essere discretizzato in modo efficae (es. pianificazione di traiettoria di un robot in un ambiente statico) in cui si può discretizzare anche $\mathbb{U}$ spazio delle decisioni.

In generale quando $\mathbb{X}$ è uno spazio ad alta dimensione si incorre nella cosiddetta curse of dimensionality, per cui la fase backward risulta computazionalmente impraticabile e si rende quindi necessaria un'approssimazione. Viene assegnata alla value function $\mathcal{V}^o(x(t), t)$ una struttura prefissata (es. una rete neurale) in cui si va a ottimizzare un vettore di parametri $w(t)$

$$\mathcal{V}^o(x(t), t) \approx \tilde{\mathcal{V}}(x(t), w(t)) \tag{6.1.14}$$



Come si fa la programmazione dinmanica approssimata su orizzone finito? Per ogni $t = T-1, \dots, 0$

1. Si genera casualmente un sottoinsieme di K stati $x_1, \dots, x_K$
2. Per ogni x_k per $k = 1, \dots, K$ si va ad applicare l'equazione della PD
$$\hat{\mathcal{V}}_k(t) = \max_{u(t)} \left\{ r(t, x_k, u(t)) + \mathbb{E}_{\xi(t)} \tilde{\mathcal{V}}(x(t+1), w(t+1)) \right\} \tag{6.1.15}$$
(supponendo, al tempo t, di avere già determinato i parametri $w(t+1)$). **N.B:** Sapendo che $\mathcal{V}^o(x(T), T) = r(x(T), T)$ questo ci consente di inizializzare la ricorsione.
3. Si risolve un problema di **regressione** determinando $w(t)$ in modo che
$$\tilde{\mathcal{V}}(x_k, w(t)) \approx \hat{\mathcal{V}}_k(t), \quad k = 1, \dots, N \tag{6.1.16}$$
in cui i dati sono le coppie (stato,valore)
$$(x_k, \hat{\mathcal{V}}_k(t)), \quad k = 1, \dots, K \tag{6.1.17}$$

Per calcolare
$$\mathbb{E}_{\xi(t)} \tilde{\mathcal{V}}(x(t+1), w(t)) \tag{6.1.18}$$
con $x(t+1) = f(x_k, u(t), \xi(t))$ si può sfruttare la pdf di Markov
$$\varphi(x(t+1)|x_k, u(t)) \tag{6.1.19}$$
ottenendo
$$\mathbb{E}_{\xi(t)} \tilde{\mathcal{V}}(x(t+1), w(t)) = \begin{cases} \int \varphi(x(t+1)|x_k, u(t)) \tilde{\mathcal{V}}(x(t+1), w(t+1)) dx(t+1), & \mathbb{X} \text{ continuo} \\ \sum_{x(t+1) \in \mathbb{X}} \varphi(x(t+1)|x_k, u(t)) \tilde{\mathcal{V}}(x(t+1), w(t+1)), & \mathbb{X} \text{ discreto} \end{cases} \tag{6.1.20}$$
Se non riusciamo a fare i calcoli in modo esatto (e di solito non ci si riesce se gli stati sono tanti) si approssima mediante metodo Monte Carlo: si generano casualmente un certo numero di stati $x_{l,k}$ secondo la pdf $\varphi(x(t+1)|x_k, u(t))$ e si va ad approssimare
$$\mathbb{E}_{\xi(t)} \tilde{\mathcal{V}}(x(t+1)|w(t)) \approx \frac{1}{M} \sum_{m=1}^{M} \tilde{\mathcal{V}}(x_{m,l}, w(t)) \tag{6.1.21}$$

Per calcolare poi la policy/legge di controllo ci sono due alternative

1. Se si riesce a fare il calcolo on-line (dipende dalla complessità del problema) possiamo porre
$$u(t) = \arg\max_{u} \left\{ r(t, x(t), u) + \mathbb{E}_{\xi(t)} \tilde{\mathcal{V}}(x(t+1), w(t+1)) \right\} \tag{6.1.22}$$
2. Se non si riesce a calcolare $u(t)$ on-line allora possiamo addestrare off-line una policy approssimata (es. rete neurale), avendo quindi
$$u(t) = \tilde{\gamma}(x(t), \theta(t)) \tag{6.1.23}$$
in cui $\theta(t)$ è il vettore dei parametri da ottimizzare per approssimare la policy ottima.

Qual è la difficoltà? una sicuramente è il **tempo**, dovendo determinare T funzioni approssimanti, una per ogni instante di tempo (se queste sono reti neurali...). La seconda difficoltà è che per determinare queste funzioni dobbiamo andare indietro nel tempo (si parte da T e si usa questa approssimazione per calcolare quella a $T-1$ e così via fino a $t+1$), in questo modo l'errore di approssimazione si propaga.

6.2 Programmazione dinamica su orizzonte infinito

upponiamo quindi di non avere un tempo finale prefissato per il problema di decisione e si considera quindi un problema su orizzonte ∞. L'obiettivo è dato da

$$\mathbb{E}_{\xi(t)}\Big\{\sum_{t=0}^{\infty} r(t, x(t), u(t))\Big\} \tag{6.2.1}$$

N.B: Non c'è il reward finale $r(x(T), T)$ dal momento che non c'e' un istante finale. Si deve garantire la convergenza della serie, una scelta tipica per garantire ciò è

$$r(t, x(t), u(t)) = \alpha^t r(x(t), u(t)) \tag{6.2.2}$$

con $\alpha \in (0, 1]$ detto *fattore di sconto*. Essendo un esponenziale tra 0 e 1 ci dice che i reward sono tanto meno importanti quanto piu sono lontani nel futuro. Matematicamente $\alpha < 1$ mi garantisce che la funzione obiettivo sia limitata quando $r(x(t), u(t))$ sono limitati, ovvero

$$\exists C \in \mathbb{R} : |r(x(t), u(t))| \leq C, \quad \forall x(t), u(t) \tag{6.2.3}$$

Il caso $\alpha = 1$ va bene solo in certi casi particolari quando con probabilità 1 arrivo in tempo finito in un insieme di stati finali *assorbenti* (da cui non posso uscire) con reward 0 (ovvero quando siamo certi che prima o poi si arriverà a una fine). Quando c'è il fattore di sconto α di solito si definisce la *value function* in modo leggermente diverso: il **reward parziale** da t in poi è dato da

$$\mathbb{E}_{\xi(s), s\geq t}\Big\{\sum_{s=t}^{\infty} \alpha^s r(x(s), u(s))\Big\} = \alpha^t \mathbb{E}_{\xi(s), s\geq t}\Big\{\sum_{s=t}^{\infty} \alpha^{s-t} r(x(s), u(s))\Big\} \tag{6.2.4}$$

ovvero che inveche che iniziare a scontare dal tempo 0 si comincia a scontare dal tempo t (al tempo $s = t$ pesa 1, a $s = t + 1$ pesa α, a $s = t + 2$ pesa α^2...) quindi il **reward potenziale ottimo** al tempo t diventa

$$\alpha^t \mathcal{V}^o(x(t), t) = \max_{u(t)} \Big\{ \underbrace{\alpha^t r(x(t), u(t))}_{\text{reward istantaneo}} + \alpha^{t+1} \mathcal{V}^o(x(t+1), t+1)\Big\} = \tag{6.2.5}$$

$$\alpha^t \mathcal{V}^o(x(t), t) = \alpha^t \max_{u(t)} \Big\{ r(x(t), u(t)) + \mathbb{E}_{\xi(t)} \alpha \mathcal{V}^o(x(t+1), t+1)\Big\} \tag{6.2.6}$$

da cui ho l'**equazione della progarmmazione dinamica con fattore di sconto**

$$\mathcal{V}^o(x(t), t) = \max_{u(t)} \Big\{ r(x(t), u(t)) + \alpha \mathbb{E}_{\xi(t)} \mathcal{V}^o(x(t+1), t+1)\Big\} \tag{6.2.7}$$

Per trovare la soluzione su orizzone **infinito** si considera il limite per $T \to \infty$ per un problema su orizzonte T con obiettivo

$$\mathbb{E}_{\xi(t)}\Big\{\sum_{t=0}^{T-1} \alpha^t r(x(t), u(t))\Big\} \tag{6.2.8}$$

Si definisce quindi la value function al tempo t per un problema su orizzonte T come $\mathcal{V}^T(x(t), t)$: tali valori sono calcolati ricorsivamente all'indietro (fase backward) per $t = T - 1, T - 2, \ldots, 0$ partendo dall'inizializzazione

$$\mathcal{V}^T(x(T), T) = 0 \tag{6.2.9}$$

Sotto opportune ipotesi (es. $\mathbb{X}$ insieme discreto) tanto piu sono lontano dall'instante finale T quanto più la value function $\mathcal{V}^T(x(t),t)$ (e di conseguenza anche la policy) tende ad essere **stazionaria** (indipendente dal tempo). Matematicamente

$$\lim_{T-t\to\infty} \mathcal{V}^T(x(t),t) = \mathcal{V}^o(x) \tag{6.2.10}$$

si converge quindi a un valore stazionario. Quindi abbiamo un'equazione, nella fase backward

$$\mathcal{V}^T(x(t),t) = \max_{u(t)} \Big\{ r(x(t),u(t)) + \alpha \mathbb{E}_{\xi(t)} \mathcal{V}^T(x(t+1),t+1) \Big\} \tag{6.2.11}$$

che converge a un punto stazionario calcolato come punto fisso della successione. Per calcolare il punto fisso si risolve

$$\mathcal{V}^T(x(t),t) = \mathcal{V}^o(x(t)), \quad \mathcal{V}^T(x(t+1),t+1) = \mathcal{V}^o(x(t+1)) \tag{6.2.12}$$

Il valore stazionario è la soluzione di

$$\mathcal{V}^o(x(t)) = \max_{u(t)} \Big\{ r(x(t),u(t)) + \alpha \mathbb{E}_{\xi(t)} \mathcal{V}^o(x(t+1)) \Big\} \tag{6.2.13}$$

che prende il nome di **equazione stazionaria di Bellman** della programmazione dinamica. La policy è anch'essa stazionaria

$$\gamma^o(x(t)) = \arg\max_{u(t)} \Big\{ r(x(t),u(t)) + \alpha \mathbb{E}_{\xi(t)} \mathcal{V}^o(x(t+1)) \Big\} \tag{6.2.14}$$

Come si calcola la soluzione stazionaria? Esistono due approcci fondamentalmente: ***value iteration*** *e* ***policy iteration***

6.3 Value Iteration

Si parte inizializzando la value function in modo arbitrario (non è necessaria l'inizializzazione a zero per la convergenza, cambia solo il tempo in cui si converge) ad esempio $\mathcal{V}_0(x) = 0$ e quindi per $k = 1, \dots$ si pone, $\forall x(t) \in \mathbb{X}$

$$\mathcal{V}_k(x(t)) = \max_{u(t)} \Big\{ r(x(t),u(t)) + \alpha \mathbb{E}_{\xi(t)} \mathcal{V}_{k-1}(x(t+1)) \Big\} \tag{6.3.1}$$

Una iterazione di *value iteration* equivale a un passo della fase backward della programmazione dinamica: invece che indicizzarla con un indice T all'indietro nel tempo si indicizza con un indice k in avanti nel tempo. Sotto opportune ipotesi, come prima (es. $\mathbb{X}$ discreto), si ha

$$k \to \infty \implies \mathcal{V}_k(x(t)) \to \mathcal{V}^o(x(t)) \tag{6.3.2}$$

convergenza al valore stazionario, soluzione dell'equazione di Bellman per la PD stazionaria. **N.B:** Quando $T \to \infty$ ogni istante finito t è infinitamente lontano dalla fine ed è questa la ragione per cui entra in gioco la stazionarietà. La **difficoltà** è legata al $\forall x(t) \in \mathbb{X}$, che per un numero di stati elevato diventa intrattabile. Quando la cardinalità è troppo alta si va ad approssimare (es. con una rete neurale)

$$\mathcal{V}^o(x(t)) \approx \tilde{\mathcal{V}}(x(t), w) \tag{6.3.3}$$

con w vettore di parametri da determinare. Ad ogni iterazione k si aggiorna il vettore w calcolando w_k in funzione di w_{k-1}: per $k = 0, 1, \dots$

1. Si generano casualmente M stati $x_1, \dots, x_M$
2. Per $m = 1, \dots, M$ si calcola $\hat{\mathcal{V}}_{k,m} = \max_{u(t)} \Big\{ r(x_m, u(t)) + \alpha \mathbb{E}_{\xi(t)} \tilde{\mathcal{V}}(x(t+1)|w_{k-1}) \Big\}$
3. Si risolve un problema di regressione determinando w_k in modo tale che l'approssimazione sia vicina al valore campionario che si è calcolato

$$\tilde{\mathcal{V}}(x_m, w_k) \approx \hat{\mathcal{V}}_{k,m}, \quad m = 1, \dots, M \tag{6.3.4}$$

sul training set $(x_m, \hat{\mathcal{V}}_{k,m})$. L'obiettivo è che per $k \to \infty$ si arriverà a convergere circa ai valori ottimi per gli stati

$$\tilde{\mathcal{V}}(x_{,} w_k) \to_{\approx} \mathcal{V}^o(x) \tag{6.3.5}$$

Nella value iterarion approssimata ad ogni iterazione (in cui si è generato un certo numero di stati, e quindi un certo numero di campioni) dobbiamo aggiornare i nostri parametri (es. ri-addestrando la rete neurale)

$$\mathcal{V}_k \approx \tilde{\mathcal{V}}(x, w_k) \tag{6.3.6}$$

Per accelerare la convergenza:

- Nella discesa del gradiente per trovare w_k posso partire da w_{k+1}
- Può essere utile considerare un termine di regolarizzazione del tipo

$$R(w) = ||w - w_{k-1}||^2 \tag{6.3.7}$$

La value iteration che abbiamo mostrato è una versione *sincrona*: esiste una versione asincrona della value iteration approsimata in cui ogni volta che si genera **uno** stato (un campione) x_k si aggiorna il vettore w_k. All'iterazione k

1. Si genera casualmente lo stato x_k
2. Si calcola $\hat{\mathcal{V}}_k = \max_{u(t)} \{ r(x_k, u(t) + \alpha \mathbb{E}_{\xi(t)} \tilde{\mathcal{V}}(x(t+1), w_{k-1}) \}$
3. Si aggiorna w_k con un passo di discesa del gradiente

$$w_k = w_{k-1} - \epsilon_k \frac{\partial}{\partial w} \frac{1}{2} \Big[\hat{\mathcal{V}}_k - \tilde{\mathcal{V}}(x_k, w) \Big]^2 \Big|_{w=w_{k-1}} = \tag{6.3.8}$$

$$= w_{k-1} + \epsilon_k \Big[\hat{\mathcal{V}}_k - \tilde{\mathcal{V}}(x_k, w_k - 1) \Big] \frac{\partial}{\partial w} \tilde{\mathcal{V}}(x_k, w_{k-1}) \tag{6.3.9}$$

N.B: Si tratta di un algoritmo di gradiente stocastico.
Può convergere più velocemente ma può avere problemi di stabilità, nella pratica si usano versioni intermedie tra la sincrona e l'asincrona.

6.4 Policy Iteration

Invece che iterare sui valori (modificando i pesi w_k) si itera rispetto alla policy: si parte da una policy iniziale $\gamma_0(x)$ e, ricorsivamente, si calcola una nuova policy migliorata $\gamma_k(x)$ a partire da quella precedente. *Considerazione:* Per valutare la bontà di una certa policy γ si considera il reward totale associato ae essa:

$$\mathcal{V}^{\gamma}(x) = \mathbb{E}_{\xi(t)}\Big\{ \sum_{t=0}^{\infty} \alpha^t r(x(t), \gamma(x(t)))\Big\}\Big|_{x(0)=x} \tag{6.4.1}$$

che può essere calcolato come soluzione del sistema di equazioni

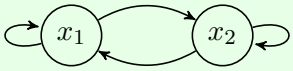
$$\mathcal{V}^{\gamma}(x(t)) = r(x(t), \gamma(x(t))) + \alpha \mathbb{E}_{\xi(t)} \mathcal{V}^{\gamma}(x(t+1)) \tag{6.4.2}$$

con $x(t+1) = f(x(t), \gamma(x(t)), \xi(t))$ per $\forall x \in \mathbb{X}$. È l'equazione della PD stazionaria, ma senza il massimo: non si sta cercando la policy ottima (per adesso) ma solamente di valutare una policy fissata γ. Di conseguenza

$$u(t) = \gamma(x(t)) \tag{6.4.3}$$

Quando $\mathbb{X}$ è un insieme discreto si tratta di risolvere un sistema di $|\mathbb{X}|$ equazioni lineari.

Esempio 6.1. *Esempio con 2 stati*



Si trovano i valori $\mathcal{V}^{\gamma}(x_1), \mathcal{V}^{\gamma}(x_2)$ risolvendo

$$\mathcal{V}^{\gamma}(x_1) = r(x_1, \gamma(x_1)) + \alpha\Big\{\varphi(x_1|x_1, \gamma(x_1))\mathcal{V}^{\gamma}(x_1) + \varphi(x_2|x_1, \gamma(x_1))\mathcal{V}^{\gamma}(x_2)\Big\} \tag{6.4.4}$$

$$\mathcal{V}^{\gamma}(x_2) = r(x_2, \gamma(x_2)) + \alpha\Big\{\varphi(x_1|x_2, \gamma(x_1))\mathcal{V}^{\gamma}(x_1) + \varphi(x_2|x_2, \gamma(x_2))\mathcal{V}^{\gamma}(x_2)\Big\} \tag{6.4.5}$$

Se si può valutare una policy allora si può anche migliorarla: si parte da una policy tentativo γ_0. Per $k = 0, 1, \ldots$

1. Si calcola $\mathcal{V}^{\gamma_k}(x)$ per ogni stato $x \in \mathbb{X}$
2. Si aggiorna la policy effettuando un passo di programmazione dinamica, $\forall x \in \mathbb{X}$

$$\gamma_{k+1}(x(t)) = \arg\max_{u(t)} \Big\{ r(x(t), u(t)) + \alpha \mathbb{E}_{\xi(t)} \mathcal{V}^{\gamma_k}(x(t+1))\Big\} \tag{6.4.6}$$

con $x(t+1) = f(x(t), u(t), \xi(t))$. In questa massimizzazione si suppone di essere nello stato $x(t)$ e si determina la migliore azione $u(t)$ supponendo che da $t+1$ in poi si applichi γ_k. Si può dimostrare che, per costruzione, poichè la $u(t)$ è ottimizzato (fissata la policy da $t+1$ in poi) allora il valore della policy che ho calcolato è sicuramente non peggiore del precedente

$$\mathcal{V}^{\gamma_{k+1}}(x(t)) \geq \mathcal{V}^{\gamma_k}(x(t)) \tag{6.4.7}$$

e quindi questo ci garantisce la convergenza dal momento che si ha una successione monotona non decrescente limitata.

$$\lim_{k\to\infty} \mathcal{V}^{\gamma_k}(x(t)) = \mathcal{V}^o(x(t)) \tag{6.4.8}$$

$$\lim_{k\to\infty} \gamma^k(x(t)) = \gamma^o(x(t)) \tag{6.4.9}$$

Osservazione: γ_0 policy iniziale va scelta mediante un'euristica/tecnica di sintesi. Ad esempio nella navigazione di robot γ_0 può essere la legge di controllo associata al metodo dei potenziali artificiali. Migliore è γ_0 tanto più rapida è la convergenza.
La policy iteration è un metodo di tipo **actor-critic**: nel passo 1 si critica la policy attuale, valutandone il reward - ossia calcolando $\mathcal{V}^{\gamma_k}(x(t))$, e nel passo 2 si agisce migliorando la policy applicando la programmazione dinamica.

Considerazione: **Come si valuta la policy?** Per calcolare la value function associata alla policy γ_k se non si riesce a risolvere in modo esatto il sistema di equazioni lineari ($\mathbb{X}$ è intrattabile) allora si utilizzano, ancora, funzioni approssimanti (es. reti neurali): si approssima la funzione di valore associata ad una certa policy fissata con una funzione approssimante, determinando il vettore dei parametri

$$\mathcal{V}^{\gamma_k}(x) \approx \tilde{\mathcal{V}}(x, w_k) \tag{6.4.10}$$

Per determinare w_k si possono applicare due metodi

1. Metodo Monte Carlo.
2. Metodo delle differenze temporali.

6.5 Metodo Monte Carlo

Data una policy fissata γ vogliamo approssimare la value function associata $\mathcal{V}^{\gamma}(x)$

1. Si generano casualmente degli M stati $x_1, \ldots, x_M$
2. Per ogni $m = 1, \ldots, M$ si **simula** l'evoluzione del sistema dinamico partendo da $x(0) = x_m$ e applicando $u(t) = \gamma(x(t))$ si calcola il corrispondente reward

$$\hat{\mathcal{V}}_m^{\gamma} = \sum_{t=0}^{\infty} \alpha^t r(x(t), \gamma(x(t)))\Big|_{x(0)=x_m} \tag{6.5.1}$$

N.B: Non si tratta di un reward approssimato perchè non si ha valore atteso ma una singola realizzazione del processo aleatorio $\{\xi(t)\}$.
N.B: Idealmente dovremmo simulare per $T \to \infty$ ma ovviamente la simulazione viene terminata ad un certo istante T^* sufficientemente grande.

3. Si determina il vettore dei parametri w della funzione approssimante risolvendo sempre un problema di regressione in modo che, per tutti gli stati che si è considerato, sia il più vicino possibile a quello calcolato

$$\tilde{\mathcal{V}}^{\gamma}(x_m, w) \approx \hat{\mathcal{V}}_m^{\gamma}, \quad m = 1, \ldots, M \tag{6.5.2}$$

N.B: Se si vuole approssimare meglio il reward vero $\mathcal{V}^\gamma(x_m)$ si devono considerare diverse traiettorie che partono da x_m associate a diverse realizzazioni di $\{\xi(t)\}$.

La **difficoltà** di questo metodo è che non è efficiente dal punto di vista del campionamento: *si simula un'intera traiettoria per calcolare un singolo valore* $\hat{\mathcal{V}}_m^\gamma$, relativo allo stato iniziale. L'ideale sarebbe sfruttare *tutti* gli stati che vengono attraversati in un traiettoria per aggiornare i parametri w. Questo è quello che viene fatto nel metodo $TD(\lambda)$.

6.6 Metodo dlle differenze temporali

Data una policy fissata γ si vuole calcolare il modo approssimato (in modo esatto usualmente non è possibile) la corrispondente value function

$$\mathcal{V}^\gamma(x) = \mathbb{E}_{\xi(t)}\Big\{ \sum_{t=0}^{\infty} \alpha^t r(x(t), u(t)) \Big\}\Big|_{x(0)=x, u(t)=\gamma(x(t))} \tag{6.6.1}$$

Ovvero il reward totale che ci aspettiamo di ottenere partendo da $x(0) = x$ associata ad una certa policy γ. La cosa importante da sottolinare è che **non partiamo dal modello**, bensì dai **dati**. Questa è una tecnica *model-free* [i.e. non richiede di conoscere la $\varphi(x(t+1)|x(t), u(t)))$]. L'idea è semplice: si supponede di essere al tempo t nello stato $x(t)$:

- Si applica $u(t) = \gamma(x(t))$
- Si ottiene un reward istantaneo $r(x(t), u(t)) \coloneqq r(t)$
- Si ha la transizione allo stato $x(t+1)$.

Supponiamo di avere già a disposizione una value function approssimata $\tilde{\mathcal{V}}^\gamma(x, w(t))$ addestrata con i dati raccolti fino al tempo t. L'obiettivo è quello di aggiornare $w(t)$ sulla base dei nuovi dati $x(t), r(t), x(t+1)$. Questo si fa attraverso un algoritmo ricorsivo, che ci permetterà di raggiungere approssimazioni sempre migliori della value function *vera*. Sulla base dei nuovi dati si ha una nuova stima del valore associato allo stato $x(t)$. La value function esatta infatti è data da

$$\mathcal{V}^\gamma(x(t)) = r(x(t), \gamma(x(t))) + \alpha \mathbb{E}_{\xi(t)} \mathcal{V}^\alpha(x(t+1)) \tag{6.6.2}$$

che non sappiamo calcolare. Approssimiamo quindi con

$$\mathcal{V}^\gamma(x(t)) \approx r(x(t), \gamma(x(t))) + \alpha \tilde{\mathcal{V}}^\alpha(x(t+1), w(t)) \tag{6.6.3}$$

In cui invece che mediando su tutte le possibili transizioni si considera solo sulla transizione effettiva che ci porta nello stato $x(t+1)$. Inoltre, al posto della value function *vera*, che stiamo cercando di calcolare e quindi non conosciamo, c'e' la sua approssimazione che conosciamo dal momento che è ottenuta dai dati. Il vettore dei parametri $w(t)$ viene aggiornato, determinando il vettore $w(t+1)$, in modo che

$$\tilde{\mathcal{V}}^\alpha(x(t), w(t+1)) \approx r(t) + \alpha \tilde{\mathcal{V}}^\gamma(x(t+1), w(t)) \tag{6.6.4}$$

In particolare si effettua un passo di discesa del gradiente

$$w(t+1) = w(t) - \beta(t) \frac{\partial}{\partial w} \frac{1}{2} \Big[r(t) + \alpha \tilde{\mathcal{V}}^\alpha(x(t+1), w(t)) - \tilde{\mathcal{V}}^\alpha(x(t), w) \Big]^2 \Big|_{w=w(t)} \tag{6.6.5}$$

in cui $\beta(t) \in (0,1)$. Si cerca di approssimata la value function nello stato $x(t)$ alla sua stima partendo dai pesi precedenti sulla base dei nuovi dati. Derivando si ha

$$w(t+1) = w(t) + \beta(t)\Big[r(t) + \alpha\tilde{\mathcal{V}}^{\gamma}(x(t+1), w(t)) - \tilde{\mathcal{V}}^{\gamma}(x(t), w(t))\Big]\Big[\frac{\partial}{\partial w}\tilde{\mathcal{V}}^{\gamma}(x(t), w)\Big]\Big|_{w=w(t)} \quad (6.6.6)$$

Che è l'algoritmo $TD(0)$. Se $\tilde{\mathcal{V}}^{\gamma}$ è una rete neurale il termine

$$\frac{\partial}{\partial w}\Big[\tilde{\mathcal{V}}^{\gamma}(x(t), w)\Big]\Big|_{w=w(t)} \quad (6.6.7)$$

si calcola con la backpropagation. Il termine

$$r(t) + \alpha\tilde{\mathcal{V}}^{\gamma}(x(t+1), w(t)) \quad (6.6.8)$$

è la nuova stima del valore dello stato $x(t)$ sulla base dei nuovi dati, mentre

$$\tilde{\mathcal{V}}^{\gamma}(x(t), w(t)) \quad (6.6.9)$$

è la stima del valore dello stato $x(t)$ sulla base dek dati fino al tempo t. La differenza tra le due

$$r(t) + \alpha\tilde{\mathcal{V}}^{\gamma}(x(t+1), w(t)) - \tilde{\mathcal{V}}^{\gamma}(x(t), w(t)) \quad (6.6.10)$$

prende il nome di **differenza temporale (time difference) al tempo** t e dà il nome all'algoritmo. L'algoritmo utilizza solo l'informazione relativa ai dati $x(t), r(t), x(t+1)$. La cosa importante è che non si deve conoscere $x(t+1) = f(x(t), u(t), \xi(t))$, la pdf di transizione di Markov e neanche la funzione di reward $r(x(t), u(t))$ ma soltanto il reward istantaneo ottenuto $r(t)$. Questo algoritmo può essere applicato on-line o anche off-line su dati registrati.

Quella che abbiamo visto ora è la versione $TD(0)$: esiste una versione più generale che è $TD(\lambda)$ in cui $\lambda \in [0,1]$

$$w(t+1) = w(t) + \beta(t)\Big[r(t) + \alpha\tilde{\mathcal{V}}(x(t+1), w(t)) - \tilde{\mathcal{V}}(x(t), w(t))\Big]\eta(t) \quad (6.6.11)$$

con

$$\eta(t) = \sum_{s=0}^{t}(\alpha\lambda)^{t-s}\frac{\partial}{\partial w}\tilde{\mathcal{V}}^{\gamma}(x(s), w(t)) \quad (6.6.12)$$

in cui per $\lambda \to 0$ si ottiene $TD(0)$, per $\lambda > 0$ si usano i nuovi dati $x(t), r(t), x(t+1)$ e quindi la differenza temportale al tempo t per aggiornare **anche** le stime dei valori degli stati attraversati in precedenza (non vediamo perchè). Si ha che quindi λ è un fattore di *dimenticanza* (forgetting factor) e per $\lambda = 1$ si ottiene un algoritmo simile al metodo Monte Carlo. Il vettore $\eta(t)$ prende il nome di *eligibility trace*.

TODO: Discorso su approssimare il valore degli stati precedenti a partire dai dati ottenuti al tempo corrente

Questo algoritmo ci permette di valutare in modo empirico la nostra policy, adesso ci interessa definire un algoritmo che permetta anche di ottimizzarla: il Q-learning.

6.7 Q-Learning

6.7.1 Versione *off-line* basata sul modello

L'obiettivo è determinare la policy ottima mediante l'algoritmo di value iteration senza richiedere di conoscere il modello di transizione di stato nè la funzione $r(x(t), u(t))$. Introduciamo la **action-value function**

$$Q^o(x(t), u(t)) = r(x(t), u(t)) + \alpha \mathbb{E}_{\xi(t)} \mathcal{V}^o(x(t+1)) \tag{6.7.1}$$

ed è la funzione che fornisce il valore (reward) dell'azione $u(t)$ nello stato $x(t)$ supponendo di applicare una generica $u(t)$ al tempo t e poi da $t+1$ in poi di applicare la policy ottima. Ricordando l'equazione di Bellman della PD stazionaria

$$\mathcal{V}^o(x(t)) = \max_{u(t)} \Big\{ r(x(t), u(t)) + \alpha \mathbb{E}_{\xi(t)} \mathcal{V}^o(x(t+1)) \Big\} \tag{6.7.2}$$

da cui si vede che

$$\mathcal{V}^o(x(t)) = \max_{u(t)} Q^o(x(t), u(t)) \tag{6.7.3}$$

per cui la policy ottima è

$$\gamma^o(x(t)) = \arg\max_{u(t)} Q^o(x(t), u(t)) \tag{6.7.4}$$

Nell'equazione di Bellman però dobbiamo conoscere il modello: se riusciamo a calcolare in qualche modo Q^o però possiamo determinare la policy ottima senza conoscere $\varphi(x(t+1)|x(t), u(t))$ nè $r(x(t), u(t))$. Nel caso $x(t) \in X, u(t) \in U$ con X, U insiemi discreti allora $Q^o(x(t), u(t))$ è una tabella di dimensione $|\mathbb{X}| \times |\mathbb{U}|$.
L'equazione della PD può essere riscritta in termini di Q^o

$$Q^o(x(t), u(t)) = r(x(t), u(t)) + \alpha \mathbb{E}_{\xi(t)} \mathcal{V}^o(x(t+1)) = \tag{6.7.5}$$

$$= r(x(t), u(t)) + \alpha \mathbb{E}_{\xi(t)} \max_{u(t+1)} Q^o(x(t+1), u(t+1)) \tag{6.7.6}$$

N.B: Rispetto all'equazione della PD si è invertito l'ordine degli operatori $\mathbb{E}$ e max.

La Q^o può essere calcolata mediante *value iteration asincrona*: ad ogni iterazione k si aggiorna il valore di una singola coppia x_k, u_k. Si parte da una $Q_0(x, u)$ iniziale e, per $k = 0, 1, \ldots$

1. Si genera una coppia (x_k, u_k).
2. Si effettua un passo di PD

$$Q_{k+1}(x_k, u_k) = r(x_k, u_k) + \alpha \mathbb{E}_{\xi(t)} \max_{u(t+1)} Q_k(x(t+1), u(t+1)) \tag{6.7.7}$$

   con $x(t+1) = f(x_k, u_k, \xi(t))$.
3. Non si modificano gli altri: $Q_{k+1}(x, u) = Q_k(x, u), \quad \forall (x, u) \neq (x_k, u_k)$

Sotto opportune ipotesi l'algoritmo converge

$$\lim_{k \to \infty} Q_k(x, u) = Q^o(x, u) \tag{6.7.8}$$

purchè tutti gi stati $(x, u) \in \mathbb{X} \times \mathbb{U}$ **siano visitati un numero *sufficiente* di volte**. Questa è una *esplorazione dello spazio degli stati e delle configurazioni.* Quello appena esposto è un algoritmo off-line che si basa sul modello ma può essere reso **model-free** e in grado di funzionare in tempo reale.

6.7.2 Versione *on-line* e *model-free*

Consideriamo il caso semplice di **transizioni deterministiche**

$$x(t+1) = f(x(t), u(t)) \tag{6.7.9}$$

Supponendo di essere al tempo t nello stato $x(t)$ e di applicare $u(t)$ si ottiene il reward $r(t)$, effettuando la transizione in $x(t+1)$. A questo punto si ha che l'iterazione k della versione precedente può essere pensata ora come il tempo

$$Q_{t+1}(x(t), u(t)) = r(t) + \alpha \max_u Q_t(x(t+1), u) \tag{6.7.10}$$

in cui $x(t+1)$ è lo stato in cui si arriva veramente al tempo $t+1$ (la transizione è osservata, non modellata). **N.B:** L'algoritmo richiede solo la conoscenza di $\{x(t)\}, \{u(t)\}$ e $\{r(t)\}$. I dati possono essere generati con qualunche policy ma *deve garantire l'esplorazione* (tutte le coppie (x, u) devono essere visitate un numero sufficiente di volte). Se non si hanno altri vincoli $u(t)$ può essere generata casualmente, altrimenti, se si deve anche cercare di ottimizzare le prestazioni mentre si impara, si può usare una strategia ϵ-greedy:

- Con probabilità ϵ si genera $u(t)$ nell'interno $\mathbb{U}$ (fase di *exploration*)
- Con probabilità $1 - \epsilon$ si genera $u(t)$ in modo da massimizzare il valore (fase di *exploitation*)

  $$u(t) = \arg\max_u Q_t(x(t), u) \tag{6.7.11}$$

  che fornisce un **compromesso** tra *eplorazione* e la *massimizzazione* del valore (**exploration vs exploitation**).

Quello del Q-learning è un algoritmo di tipo off-policy perchè la policy applicata per generare i dati può anche essere indipendente da $Q_t(x(t), u(t))$.

Nel caso di **transizioni stocastiche**

$$x(t+1) = f(x(t), u(t), \xi(t)) \iff \varphi(x(t+1)|x(t), u(t)) \tag{6.7.12}$$

non si può applicare la value iteration esatta sulla base dei dati

$$Q_{t+1}(x(t), u(t)) = r(t) + \alpha \mathbb{E}_{\xi(t)} \max_u Q_t(x(t+1), u) \tag{6.7.13}$$

perchè $x(t+1)$ è unico (quello generato nel sistema reale) e quindi non si può calcolare il valore atteso rispetto a tutte le possibili transizioni: si può soltanto approssimare

$$\mathbb{E}_{\xi(t)} \max_u Q_t(x(t+1), u) \approx \max_u Q_t(x(t+1), u) \tag{6.7.14}$$

massimizzando rispetto all'unica transizione realmente osservata. La stima $Q_{t+1}(x(t), u(t))$ viene quindi modificata come una media pesata

$$Q_{t+1}(x(t), u(t)) = (1 - \beta(t))\Big[Q_t(x(t), u(t))\Big] + \beta(t)\Big[r(t) + \alpha \max_u Q_t(x(t+1), u)\Big] \tag{6.7.15}$$

tra la stima precedente (pesata con $1 - \beta(t)$) e la nuova stima del valore della coppia $x(t), u(t)$ con $\beta(t) \in [0, 1]$, che prende il nome di *learning step* (tipicamente $\beta(t) \to 0$ per $t \to \infty$).
L'algoritmo di Q-learning appena esaminato può anche essere scritto come

$$Q_{t+1}(x(t), u(t)) = Q_t(x(t), u(t)) + \beta(t)\Big[r(t) + \alpha \max_u Q_t(x(t+1, u)) - Q_t(x(t), u(t))\Big] \tag{6.7.16}$$

in cui la parte tra parentesi quadre è la differenza temporale al tempo t. Si aggiorna quindi la Q_t come ne metodo $TD(0)$ (che però è un metodo on policy) sulla base della differenza temporale (in cui però compare un max). L'algorimo converge se tutte le coppie x, u sono visitate *infinite* volte.

6.7.3 Q-Learning approssimato

Ricordiamo che la tecnica di Q-learning 6.7.15 va bene quando $\mathbb{X} \times \mathbb{U}$ è un insieme di cardinalità trattabile, altrimenti si va ad utilizzare una funzione approssimante

$$Q^o(x(t), u(t)) \approx \tilde{Q}(x(t), u(t), w) \tag{6.7.17}$$

L'**idea** è che al tempo t si aggiornano i parametri $w(t)$ sulla base di $x(t), u(t), r(t), x(t+1)$ mediante un passo di discesa del gradiente:

$$w(t+1) = w(t) - \beta(t)\frac{\partial}{\partial w}\frac{1}{2}\Big[r(t) + \alpha \max_u \tilde{Q}(x(t+1), u, w(t)) - \tilde{Q}(x(t), u(t), w)\Big]^2\Big|_{w=w(t)} \tag{6.7.18}$$

dove il primo termine nelle parentesi quadre è la nuova stima di $Q(x(t), u(t))$. Facendo la derivata si vede bene il rapporto con il Q-learning standard

$$\begin{aligned} &w(t+1) = \\ &= w(t) + \beta(t)\Big[r(t) + \alpha \max_u \tilde{Q}(x(t+1), u, w(t)) - \tilde{Q}(x(t), u(t), w(t))\frac{\partial}{\partial w}\Big]\tilde{Q}(x(t), u(t), w(t)) \end{aligned}$$

Questa espressione è analoga al metodo $TD(0)$ con $\mathcal{V}$ sostituita da Q e con il max dal momento che stiamo cercando la Q^o. La quantità tra parentesi è la differenza temporale al tempo t. Questo metodo va bene quando $\tilde{Q}(x, u, w)$ dipende **linearmente** da w, se invece la struttura è più complicata (es. reti neurali) si hanno due problemi

1. Il target della rete
$$r(t) + \alpha \max_u \tilde{Q}(x(t+1), u, w(t)) \tag{6.7.19}$$
dipende dall'attuale vettore di parametri $w(t)$: aggiornando la Q ad ogni istante temporale si va a modificare anche il target della rete, che può dare fastidio: introducendo la ricorsione nell'aggiornamento dei w si può andare incontro a **fenomeni di oscillazione/instabilità**.

2. L'ordine con cui arrivano i dati non è completamente casuale perchè $x(t), u(t), r(t), x(t+1) \to x(t+1), u(t+1), r(t+1), x(t+2)$, ovvero i dati successivi sono fortemente correlati tra loro, andando a **perdere le buone proprietà di convergenza del gradiente stocastico**.

Per risolvere il primo si utilizzano **due** funzioni approssimanti $\tilde{Q}(x(t), u(t), w(t))$ e $\tilde{Q}(x(t), u(t), \hat{w}(t))$ dove la seconda prende il nome di target network è utilizzata per il calcolo del target, che diventa

$$r(t) + \alpha \max_u \tilde{Q}(x(t+1), u, \hat{w}(t)) \tag{6.7.20}$$

Periodicamente, ogni T istanti, si pone $\hat{w}(t) = w(t)$.

Per risolvere il secondo si fa uso di una tecnica di *experience replay*: non si utilizzano i dati nell'ordine in cui arrivano (essendo un algoritmo off-policy si può fare). Si tengono in memoria tutti i dati $x(s), u(s), r(s), x(s+1)$ per $s = 0, \dots, t-1$ e, ad ogni istante t:

1. Si aggiunge $x(t), u(t), r(t), x(t+1)$ alla memoria.
2. Si estraggono casualmente K istanti temporali.
3. Per $k = 1, \dots, K$ si calcola la stima aggiornata della Q^o

$$\hat{Q}_{t,k} = r(t_k) + \alpha \max_u \tilde{Q}(x(t_{k+1}), u, \hat{w}(t)) \tag{6.7.21}$$

4. Si effettua un passo di discesa del gradiente in modo che

$$\tilde{Q}(x(t_k), u(t_k), w(t+1)) \approx \hat{Q}_{t,k} \tag{6.7.22}$$

   ovvero (prendendo una loss quadratica)

$$w(t+1) = w(t) - \beta(t) \frac{\partial}{\partial w} \frac{1}{2} \sum_{k=1}^{K} \Big[\hat{Q}_{t,k} - \tilde{Q}(x_k), u(t_k), w) \Big]^2 \Big|_{w=w(t)} \tag{6.7.23}$$

5. Se t è multiplo di T si pone $\tilde{w}(t) = w(t+1)$.

6.8 Cenni al Reinforcement Learning Multi-Agente

Consideriamo N agenti indipendenti

$$x(t+1) = f(x(t), u_1(t), \dots, u_N(t), \xi(t)) \tag{6.8.1}$$

in cui

- Ogni agente osserva lo stato $x(t)$ o una sua parte $x_i(t)$
- Decide quale controllo/azione applicare $u_i(t) = \gamma_i(x_i(t))$ in modo da massimizzare un obiettivo

$$\mathbb{E}_{\xi(t)} \Big\{ \sum_{t=0}^{\infty} \alpha^t r_i(x(t), u_1, \dots, u_N(t)) \Big\} \tag{6.8.2}$$

  in cui si distinguono

  1. Caso cooperativo $r_i = r, \quad \forall i = 1, \dots, N$
  2. Caso competitivo, es. $N = 2$ con $r_1 = -r_2$

3. Caso misto

Per ottenere γ_i invece possiamo adottare

- Un approccio **globale** in cui, per ogni agente, si impara una Q del tipo

$$Q_i(x_i, u_1, \dots, u_N) \tag{6.8.3}$$

che dipende anche dalle decisioni degli altri agenti. Questo approccio ha il problema della complessità crescente in modo esponenziale con il numero N di agenti. Si ha inoltre necessità di un **modello di decisione** per gli altri agenti, dal momento che si coosce solo x_i, u_i e gli altri u no. **Si può fare in casi particolari**:
Es. nel caso cooperativo a informazione completa $r_i = r$ e $x_i = x$ per ogni i:

$$\begin{aligned} &Q_{i,t+1}(x(t), u_1(t), \dots, u_N(t)) = \\ &= [1-\beta(t)]Q_{i,t}(x(t), u_1, \dots, u_N(t)) + \beta(t)\Big[r(t) + \alpha \max_{u_i} \max_{u_j, j\neq i} Q_{i,t}(x(t+1), u_1, \dots, u_N)\Big] \end{aligned}$$

dove il $\max_{u_j, j\neq i}$ si ha perche l'obiettivo è comune e si può supporre che anche gli altri agenti massimizzino la stessa Q. **N.B:** Se gli agenti utilizzano gli stessi dati allora $Q_i = Q$ per ogni i dal momento che stanno imparando la stessa Q può richiedere centralizzazione o coordinamento nel caso distribuito. Es. caso competitivo in cui $N = 2$ e $r_1 = -r_2$. Si ha

$$\begin{aligned} &Q_{i,t+1}(x(t), u_1(t), u_2(t)) = \\ &= [1-\beta(t)]Q_{1,t}(x(t), u_1(t), u_2(t)) + \beta(t)[r_1(t) + \alpha \max_{u_1} \min_{u_2} Q_{1,t}(x(t+1), u_1, u_2)] \end{aligned}$$

che è un approccio *minimax* in cui si ottimizza il caso pessimo rispetto alle decisioni dell'avversario. **N.B:** Si suppone che l'altro agente $Q_{2,t} = -Q_{1,t}$ (che potrebbe non essere vero).

- Un approccio **locale**: Independent learning con loss quadratica. si considerano gli altri agenti come parte dell'ambiente. Si impara una Q_i del tipo

$$Q_i(x_i(t), u_i(t)) \tag{6.8.4}$$

N.B: Si può fare perchè il Q-learning è model-free, da cui

$$Q_{i,t+1}(x_i(t), u_i(t)) = [1-\beta(t)]Q_{i,t}(x_i(t), u_i(t)) + \tag{6.8.5}$$

$$\beta(t)[r_i(t) + \alpha \max_{u_i} Q_{i,t}(x_i(t+1), u_i)] \tag{6.8.6}$$

Si deve conoscere semplicemente, per ogni agente i, $x_i(t), u_i(t), r_i(t), x_i(t+1)$. La difficoltà qui sta nel: se anche gli altri agenti stanno imparando l'ambiente generalizzato formato da ambiente+altri agenti **non** è più **stazionario** ma dipende da t. Esempio

$$x(t+1) = f(x(t), u_1(t), u_2(t), \xi(t)) \tag{6.8.7}$$

$$u_2(t) = \gamma_2(x(t)) \tag{6.8.8}$$

con policy γ_2 fissata per l'agente 1 l'ambiente generalizzato è stazionario

$$x(t+1) = f(x(t), u_1(t), \gamma(x(t), \xi(t))) \tag{6.8.9}$$

Se invece

$$u_2(t) = \gamma_2(x(t), t) \tag{6.8.10}$$

è una policy tempo-variante perche anche l'agente 2 sta imparando allora

$$x(t+1) = f(x(t), u_1(t), \gamma_2(x(t), t), \xi(t)) \tag{6.8.11}$$

si ha un contributo non stazionario e la convergenza del Q-learning non è più garantita (si deve dimenticare il passato in cui gli altri agenti stavano ancora imparando per avere un ambiente stazionario, ad es. si deve stare attenti nell'experience replay perchè i dati possono essere non più rappresentativi). Inoltre in un contesto non stazionario il learning rate β **non deve andare a zero**. In alternativa, quando possibile, impara solo un agente alla volta e gli altri agenti hanno una policy fissata e poi si alternano gli agenti che imparano. Quando l'ambiente è stazionario si parla solamente di problema di apprendimento, quando l'ambiente è non stazionario si parla di problema di addestramento.