Ricerca Operativa

- **Ricerca Operativa**
  - **Programmazione Matematica**
  - **Programmazione Lineare (PL)**
  - **Programmazione Lineare Intera (PLI)**
  - **Modelli di Programmazione Lineare Intera (PLI)**
  - **Constraint Programming (CP)**
  - **Teoria dei Grafi**
  - **Cammini Minimi (Shortest Paths)**
  - **Alberi Ricoprenti (Spanning Trees)**
  - **Problema del Commesso Viaggiatore (TSP)**
  - **Problema del Commesso Viaggiatore Asimmetrico (ATSP)**
  - **Problema della Cricca Massima (Maximum Clique Problem)**
  - **Problema della Quasi-Cricca Massima (Maximum Quasi-Clique Problem)**
  - **Problema del Massimo Insieme Indipendente (Maximum Independent Set Problem)**
  - **Problema della Colorazione dei Grafi (Graph Coloring Problem)**
  - **Problema di Pianificazione di Progetti con Vincoli di Risorse (RCPSP)**

Programmazione Matematica

Introduzione alla Programmazione Matematica

La **Programmazione Matematica** è una disciplina che si occupa di ottimizzare (minimizzare o massimizzare) una funzione obiettivo, soggetta a vincoli. È ampiamente utilizzata in ambiti come la logistica, la produzione, la finanza e molti altri.

Ecco una panoramica della notazione utilizzata:

- **Insiemi**:
  - $\mathbb{R}$: Insieme dei numeri reali.
  - $\mathbb{R}^n$: Spazio vettoriale a n dimensioni.
  - $\mathbb{Z}$: Insieme dei numeri interi.
  - $\mathbb{Z}^+$: Numeri interi positivi.
  - $[a, b]$: Intervallo chiuso $\{x \in \mathbb{R} : a \leq x \leq b\}$.
  - (a, b): Intervallo aperto $\{x \in \mathbb{R} : a < x < b\}$.
- **Norme e Valori Assoluti**:
  - $\|x\| = \sqrt{\sum_{i=1}^{n} x_i^2}$: Norma euclidea di un vettore x.
  - $|z|$: Valore assoluto di uno scalare z.
- **Insiemi e Cardinalità**:
  - $Q = \{q_1, \ldots, q_n\}$: Insieme di n elementi.
  - $Q = \{x \in \mathbb{R}^n : P(x)\}$: Insieme dei punti che soddisfano le condizioni P.
  - $|Q|$: Cardinalità dell'insieme Q.
- **Funzioni e Operatori**:
  - $\arg\min\{f(i) : i \in I\}$: Punto $i^* \in I$ che minimizza $f(i)$.
  - $\lfloor z \rfloor$: Parte intera inferiore di z.
  - $\lceil z \rceil$: Parte intera superiore di z.

Vettori e Matrici

- **Vettori**:
  - $x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$: Vettore colonna n-dimensionale.
  - $c^T = [c_1, \ldots, c_n]$: Vettore riga n-dimensionale.
  - $c^T x = \sum_{j=1}^{n} c_j x_j$: Prodotto scalare.

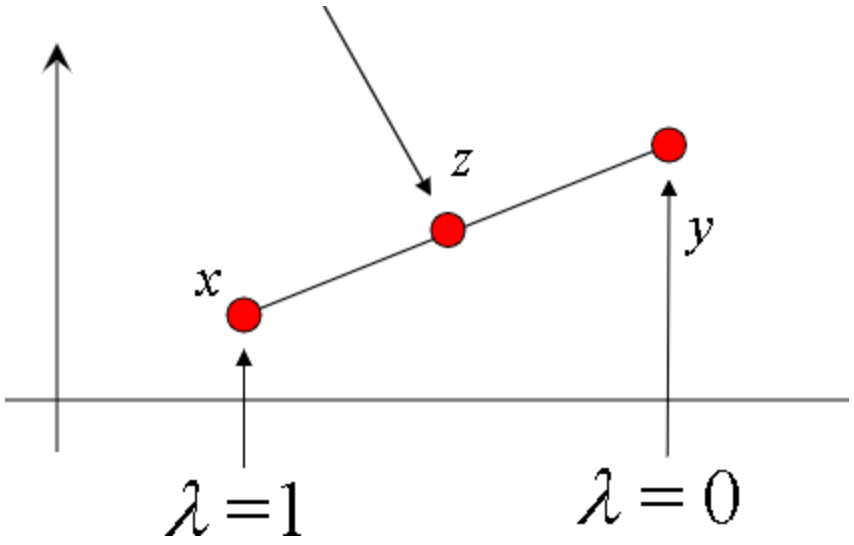
- **Matrici**:
  - $A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{pmatrix}$: Matrice $m \times n$.
  - $Ax = \begin{pmatrix} \sum_{j=1}^{n} a_{1j} x_j \\ \vdots \\ \sum_{j=1}^{n} a_{mj} x_j \end{pmatrix}$: Prodotto matrice-vettore.
  - $\text{rango}(A)$: Rango della matrice A.
  - $\det(A)$: Determinante di A.
  - A^{-1}: Matrice inversa di A.

Combinazione Convessa

- **Definizione**:
  - Un punto z è una **combinazione convessa** di x e y se esiste $\lambda \in [0, 1]$ tale che:

$$z = \lambda x + (1 - \lambda) y$$

  - Esempio: Se $x, y \in \mathbb{R}^2$, z è un punto sul segmento che congiunge x e y.



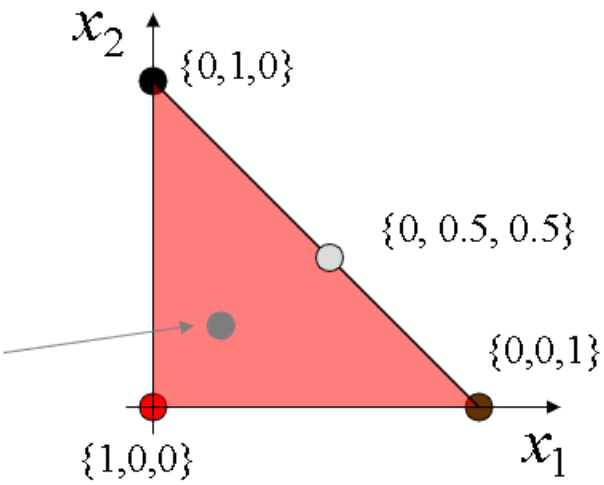
- **Combinazione Convessa di K punti**:
  - Dati $p_1, p_2, \dots, p_K \in \mathbb{R}^n$, z è una combinazione convessa se:

$$z = \sum_{i=1}^{K} \lambda_i p_i \quad \text{con} \quad \lambda_i \geq 0 \quad \text{e} \quad \sum_{i=1}^{K} \lambda_i = 1$$

Es. $z = \lambda x + (1-\lambda) y$, $\lambda_1 = \lambda > 0$, $\lambda_2 = 1-\lambda$, $\lambda_1 + \lambda_2 = 1$

$$p_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad p_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad p_3 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\lambda_i = \{0.5, 0.2, 0.3\} \quad z = \begin{bmatrix} 0.2 \\ 0.3 \end{bmatrix}$$



Insiemi Convessi

- **Definizione**:
  - Un insieme $F \subseteq \mathbb{R}^n$ è **convesso** se, per ogni $x, y \in F$ e $\lambda \in [0,1]$, si ha:

$$z = \lambda x + (1-\lambda) y \in F$$

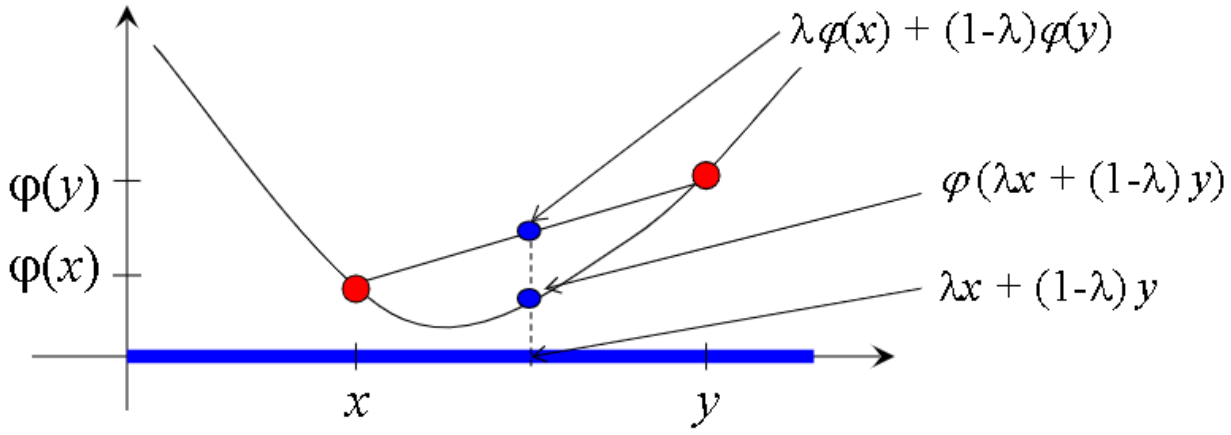
  - Esempi:
    - Un cerchio è un insieme convesso.
    - Una stella non è un insieme convesso.
- **Proprietà**:
  - L'intersezione di insiemi convessi è convessa.
  - $\mathbb{R}^n$ è convesso.

Funzioni Convesse

- **Definizione**:
  - Una funzione $\phi : F \to \mathbb{R}$ è **convessa** se, per ogni $x, y \in F$ e $\lambda \in [0,1]$, si ha:

$$\phi(\lambda x + (1-\lambda) y) \leq \lambda \phi(x) + (1-\lambda) \phi(y)$$

Es. $F = [0,1] \subset \mathrm{R}$



Problemi di Ottimizzazione

- **Formulazione Generale**:
  - Dato un vettore di variabili decisionali $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, un insieme ammissibile $F \subseteq \mathbb{R}^n$ e una funzione obiettivo $\phi : F \to \mathbb{R}$, il problema di ottimizzazione è:

$$\min \phi(x) \quad \text{con} \quad x \in F$$

  - L'obiettivo è trovare $x^* \in F$ (ottimo globale) tale che:

$$\phi(x^*) \leq \phi(x) \quad \forall x \in F$$

- **Regione Ammissibile**:
  - F può essere definita esplicitamente (es. $[0,1]^2$) o implicitamente tramite vincoli (es. $5x_1 + 3x_2 \leq 15$).

Minimi Locali e Globali

- **Definizione**:
  - Un punto $y \in F$ è un **minimo locale** se esiste un intorno $N \subseteq F$ tale che:

$$\phi(y) \leq \phi(x) \quad \forall x \in N$$

  - Un **minimo globale** è un punto $x^* \in F$ tale che:

$$\phi(x^*) \leq \phi(x) \quad \forall x \in F$$

- **Convessità e Ottimalità**:
  - Se F è convesso e ϕ è convessa, ogni minimo locale è anche un minimo globale.

Classificazione dei Problemi di Ottimizzazione

- **Programmazione Lineare (PL)**:
  - ϕ, g_i, h_j sono funzioni lineari.
  - Esistono algoritmi efficienti (es. Simplesso).
- **Programmazione Lineare Intera (PLI)**:
  - Variabili decisionali intere.
  - Problema difficile, risolto con algoritmi come Branch-and-Bound.
- **Programmazione Non Lineare (PNL)**:
  - ϕ, g_i, h_j sono funzioni non lineari.
  - Non esistono algoritmi generali, ma metodi per trovare ottimi locali.
- **Programmazione Convessa (PC)**:
  - ϕ e g_i sono convesse, h_j sono lineari.
  - Ottimo locale $\equiv$ ottimo globale.

Algoritmi Numerici

- **Iterazione**:
  - Gli algoritmi di ottimizzazione sono generalmente iterativi:
    a. Partono da una soluzione iniziale x_0.
    b. Generano una sequenza $x_0, x_1, \ldots, x_k$ che converge a x^*.
    c. Terminano quando x_k è ottima o si raggiunge una condizione di terminazione.
- **Convergenza**:
  - Nel caso generale, la convergenza è a un ottimo locale.
  - Nel caso convesso, la convergenza è a un ottimo globale.

Esempi di Problemi di Ottimizzazione

- **Esempio 1**:
  - Problema continuo: $F = [0, 3/2]^2, \max \phi(x) = x_1 + x_2$.
  - Soluzione: $x^* = (3/2, 3/2)$.
- **Esempio 2**:
  - Problema discreto: $F = [0, 3/2]^2 \cap \mathbb{Z}^2, \max \phi(x) = x_1 + x_2$.
  - Soluzione: $x^* = (1, 1)$.
- **Esempio 3**:
  - Problema continuo: $F = [0, 1]^2, \min \phi(x) = (x_1 - 1/2)^2 + (x_2 - 1/2)^2$.
  - Soluzione: $x^* = (1/2, 1/2)$.

Programmazione Lineare (PL)

Introduzione alla Programmazione Lineare

La **Programmazione Lineare (PL)** è una tecnica di ottimizzazione utilizzata per massimizzare o minimizzare una funzione obiettivo lineare, soggetta a vincoli lineari. È ampiamente applicata in problemi di produzione, logistica, gestione delle risorse e altro.

Un problema di PL è definito come:

- **Funzione obiettivo lineare**: $\phi(x) = c^T x = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n$.
- **Regione ammissibile** $F \subseteq \mathbb{R}^n$: Definito da vincoli lineari $g_i(x) \leq 0$ e $h_j(x) = 0$, dove g_i e h_j sono funzioni lineari.

Forma matriciale:

$$\min c^T x \quad \text{s.t.} \quad Ax \geq d, \quad x \geq 0$$

- A: Matrice dei coefficienti dei vincoli.
- d: Vettore dei termini noti.
- c: Vettore dei coefficienti della funzione obiettivo.

Regione Ammissibile e Convessità

- **Regione ammissibile** F: È un **poliedro convesso**, ovvero un insieme convesso definito dall'intersezione di un numero finito di semispazi.
- **Vertici**: La soluzione ottima di un problema di PL si trova sempre in uno dei vertici del poliedro F.

Formulazione di Problemi di PL
La formulazione di un problema di PL segue questi passi:

1. **Identificare le variabili decisionali**.
2. **Definire la funzione obiettivo**.
3. **Definire i vincoli**.
4. **Aggiungere vincoli di non negatività**.

Esempi Pratici di PL

Esempio 1: Produzione di Sedie

Descrizione del Problema
Un'azienda produce due tipi di sedie: **Sedia in Legno (SL)** e **Sedia in Alluminio (SA)**. La produzione utilizza tre reparti:

- **Lavorazione parti in Legno (RL)**: dedicato alla lavorazione delle parti in legno.
- **Lavorazione parti in Alluminio (RA)**: dedicato alla lavorazione delle parti in alluminio.
- **Lavorazione parti in Tessuto (RT)**: dedicato al rivestimento delle sedie.

Dati di Produzione

- **Tempi di produzione (minuti per pezzo)**:
  - **SL**: RL = 10, RT = 30.
  - **SA**: RA = 20, RT = 20.
- **Ricavo netto (euro per pezzo)**:
  - **SL**: 30 €/pezzo.
  - **SA**: 50 €/pezzo.
- **Disponibilità dei reparti (minuti per periodo)**:
  - RL: 40 minuti.
  - RA: 120 minuti.
  - RT: 180 minuti.

Formulazione del Problema

Variabili decisionali:

- x_1: Numero di sedie in legno prodotte in un periodo.
- x_2: Numero di sedie in alluminio prodotte in un periodo.

Funzione obiettivo:
Massimizzare il ricavo totale:

$$\max z = 30x_1 + 50x_2$$

Vincoli:

1. Lavorazione parti in legno (RL):

$$10x_1 \leq 40$$

2. Lavorazione parti in alluminio (RA):

$$20x_2 \leq 120$$

3. Lavorazione parti in tessuto (RT):

$$30x_1 + 20x_2 \leq 180$$

4. Vincoli di non negatività:

$$x_1 \geq 0,\ x_2 \geq 0$$

Modello Lineare Completo

$$\max z = 30x_1 + 50x_2$$

soggetto a:

$$10x_1 \leq 40 \quad (\text{RL})$$
$$20x_2 \leq 120 \quad (\text{RA})$$
$$30x_1 + 20x_2 \leq 180 \quad (\text{RT})$$
$$x_1, x_2 \geq 0$$

Esempio 2: Produzione di Vasche

- **Variabili decisionali**:
  - x_1: Numero di vasche Blue Tornado.
  - x_2: Numero di vasche Hot Spring.
- **Funzione obiettivo**:

$$\max z = 350x_1 + 300x_2$$

- **Vincoli**:
  - $x_1 + x_2 \leq 200$ (Motori disponibili).
  - $9x_1 + 6x_2 \leq 1566$ (Ore di lavoro).
  - $12x_1 + 16x_2 \leq 2880$ (Metri di tubazione).
  - $x_1, x_2 \geq 0$.

Esempio 3: Problema della Dieta

- **Variabili decisionali**:
  - x_1, x_2, x_3: Quantità di alimenti A1, A2, A3 da acquistare.
- **Funzione obiettivo**:

$$\min z = 5x_1 + 2x_2 + x_3$$

- **Vincoli**:
  - $5x_1 + 3x_2 + 3x_3 \geq 8$ (Carboidrati).
  - $3x_1 + 3x_2 + x_3 \geq 4$ (Grassi).
  - $x_2 + 2x_3 \geq 20$ (Proteine).
  - $x_1, x_2, x_3 \geq 0$.

Soluzione Grafica di Problemi di PL

Per problemi con 2 variabili, è possibile risolvere graficamente:

1. **Disegnare i vincoli** per definire la regione ammissibile.

2. **Identificare i vertici** della regione ammissibile.
3. **Valutare la funzione obiettivo** in ciascun vertice per trovare l'ottimo.

Esempio:

$$\max z = 3x_1 + 5x_2$$

soggetto a:

$$x_1 \leq 4, \quad 2x_2 \leq 12, \quad 3x_1 + 2x_2 \leq 18, \quad x_1, x_2 \geq 0$$

- **Vertici**: $(0,0), (4,0), (4,3), (2,6), (0,6)$.
- **Soluzione ottima**: $(2,6)$ con $z = 36$.

Forme di Programmazione Lineare

- **Forma generale**:

$$\min c^T x \quad \text{s.t.} \quad Ax \geq d, \quad x \geq 0$$

- **Forma canonica**:

$$\min c^T x \quad \text{s.t.} \quad Ax \geq d, \quad x \geq 0$$

- **Forma standard**:

$$\min c^T x \quad \text{s.t.} \quad Ax = d, \quad x \geq 0$$

Equivalenza tra forme:

- Le tre forme sono equivalenti e possono essere trasformate l'una nell'altra tramite:
  - Aggiunta di variabili slack/surplus.
  - Sostituzione di variabili libere con differenze di variabili non negative.

Teoremi Fondamentali

- **Teorema 1**: Ogni punto di un politopo è una combinazione convessa dei suoi vertici.
- **Teorema 2**: In un problema di PL con F non vuoto e limitato, esiste sempre almeno un vertice ottimo.

Limiti della Programmazione Lineare

- **Proporzionalità**: L'effetto delle variabili è proporzionale al loro valore.
- **Additività**: Non ci sono interazioni tra le variabili.

- **Divisibilità**: Le variabili possono assumere valori non interi.
- **Certezza**: I parametri sono noti e costanti.

Spreadsheet e Online LP Solvers

Risoluzione di Problemi di PL con più di 2 Variabili

- **Caso 3D (3 Variabili)**:
  - Un **vertice** in 3D è definito da 3 equazioni/disequazioni.
  - **Proprietà dei vertici**: La soluzione ottima si trova ancora in un vertice della regione ammissibile.
  - **Complessità**: Per un problema con m vincoli, ci sono $\binom{m}{3}$ potenziali vertici.
  - **Condizione aggiuntiva**: Il punto deve essere **ammissibile** (verificare tutti i vincoli rimanenti).

Esercizio:
Rappresentare graficamente la regione ammissibile del seguente problema di PL:

$$\max Z = X + Y + Z$$

soggetto a:

$$2X + 4Y \leq 4, \quad X, Y, Z \geq 0$$

Soluzione di Problemi di PL con Software

- **Excel Solver**:
  - Basato sul **metodo del Simplesso**, una versione algebrica del metodo grafico.
  - Valuta una sequenza di vertici rilevanti per trovare la soluzione ottima.
  - Fornisce anche informazioni utili per l'analisi di sensitività.
  - **Limiti**: Supporta fino a 200 variabili decisionali e 100 vincoli.
- **Software Specializzati**:
  - Utilizzati in ambito industriale per risolvere problemi di PL e altri modelli analitici.
  - Esempi: IBM ILOG CPLEX, GUROBI, FICO XPRESS.
  - Alcuni sono gratuiti per ricerca e educazione (es. Soplex, COIN-OR, QSopt).

3. Precisione Numerica in Excel

- **Problemi di precisione**:
  - I tool di calcolo, incluso Excel, hanno una precisione numerica limitata.
  - Problemi sorgono quando:
    - Il modello non è formulato correttamente.

        - Il problema è molto grande (molte variabili e vincoli).
        - I coefficienti delle variabili sono molto grandi o molto piccoli.
        - I dati di input sono complicati (valori molto piccoli/grandi).
    - La precisione può essere regolata, ma maggiore precisione richiede più tempo di esecuzione.

Ambiguità del Solver

- **Esempio**:

$$\max Z = X + Y$$

  soggetto a:

$$X \leq 2, \quad Y \leq 3, \quad X + Y \leq 4, \quad X, Y \geq 0$$

    - La soluzione ottima può variare in base a:
        - Formulazione del problema.
        - Ordine delle variabili e delle espressioni inserite nel solver.
        - Parametri del solver (es. metodo di soluzione, tolleranze numeriche).

Limiti di Excel

- **Dimensioni massime**:
    - Fino a 200 variabili decisionali.
    - Fino a 100 vincoli.
- **Complessità**:
    - Esistono problemi di PL più difficili da risolvere rispetto ad altri.
    - La complessità pratica non è costante e dipende dalla struttura del problema.

OpenSolver

- **Cos'è OpenSolver**:
    - Un add-in per Excel che estende le funzionalità del Solver integrato.
    - Utilizza il motore di ottimizzazione **COIN-OR CBC** per risolvere problemi di PL (inclusi quelli interi) in modo efficiente.
    - Compatibile con i modelli esistenti di Excel Solver.
    - **Nessun limite** sulla dimensione del problema.
- **Funzionalità aggiuntive**:
    - Visualizzazione del modello direttamente sul foglio di calcolo.
    - Modalità **QuickSolve** per risolvere rapidamente il modello dopo modifiche.
    - Strumento di costruzione automatica del modello.

- **Installazione**:
  i. Scaricare `OpenSolver.zip` da www.opensolver.org.
  ii. Estrarre i file in una cartella.
  iii. Aprire `OpenSolver.xlam` in Excel.
  iv. I comandi di OpenSolver appariranno nella scheda **Dati** di Excel.

Online Solvers

- **Strumenti online per risolvere problemi di PL**:
  - Zweigmedia Simplex
  - PHPSimplex
  - LP Tools
  - Online Optimizer
  - Mathstools Simplex Calculator
  - WolframAlpha LP Solver
  - Desmos Calculator (utile per rappresentare graficamente le regioni ammissibili).

Programmazione Lineare Intera (PLI)

Introduzione alla Programmazione Lineare Intera (PLI)

La **Programmazione Lineare Intera (PLI)** è una variante della Programmazione Lineare (PL) in cui le variabili decisionali devono assumere valori **interi**. Questo vincolo aggiuntivo rende il problema più complesso da risolvere rispetto alla PL continua.

- **Formulazione generale**:

$$z_P = \min c^T x \quad \text{s.t.} \quad Ax \geq d, \quad x \geq 0, \quad x \in \mathbb{Z}$$

  - $x \in \mathbb{Z}$: Vincolo di interezza (non lineare).
  - x binaria: $x(x-1) = 0$ (caso particolare di interezza).

Rilassamento Continuo di PLI

- **Rilassamento continuo $C(P)$**:
  - Si rimuove il vincolo di interezza, trasformando il problema in una PL continua.
  - La soluzione del rilassamento continuo $z_{C(P)}$ è un **limite inferiore** per la soluzione ottima del problema intero z_P:

$$z_{C(P)} \leq z_P$$

- Se la soluzione del rilassamento continuo $x_{C(P)}$ è intera, allora è anche ottima per il problema intero P.

Problemi e Algoritmi per PLI

- **Algoritmi esatti**:
  - Determinano la soluzione ottima, ma il tempo di calcolo cresce esponenzialmente con la dimensione del problema.
  - Esempi: Branch-and-Bound, Cutting Planes, Programmazione Dinamica.
- **Algoritmi euristici**:
  - Forniscono soluzioni ammissibili di buona qualità in tempi ragionevoli.
  - Non garantiscono l'ottimalità, ma possono essere utili per problemi di grandi dimensioni.

Algoritmo Euristico per PLI
Un semplice algoritmo euristico per PLI è il seguente:

1. Risolvi il rilassamento continuo $C(P)$ con il metodo del Simplesso.
2. Se $C(P)$ è impossibile, allora P è impossibile.
3. Se $C(P)$ è illimitato, allora P è illimitato (tranne casi particolari).
4. Se la soluzione $x_{C(P)}$ è intera, allora è ottima per P.
5. Altrimenti, arrotonda ogni componente frazionaria di $x_{C(P)}$ all'intero più vicino.

Problemi:

- L'arrotondamento può produrre soluzioni non ammissibili o di scarsa qualità, specialmente quando i valori delle variabili sono piccoli.

Casi di Applicazione

- **Caso 1: Soluzioni utili**:
  - Quando i valori delle variabili sono molto grandi, l'arrotondamento può essere efficace.
  - Esempio: Produzione di grandi quantità di pezzi.

$$x_1 = 2449.51 \to 2450, \quad x_2 = 14301.1 \to 14301$$

- **Caso 2: Soluzioni inutili**:
  - Quando i valori delle variabili sono piccoli, l'arrotondamento può portare a soluzioni non ammissibili o di scarsa qualità.
  - Esempio: Numero di edifici da costruire, veicoli da assegnare.
- **Caso 3: Soluzioni non ammissibili**:
  - L'arrotondamento può produrre soluzioni che violano i vincoli.
  - Esempio: Nessuno dei punti interi vicini alla soluzione continua è ammissibile.

Formulazioni Equivalenti

- **Formulazioni di PLI**:
  - Diverse formulazioni possono rappresentare lo stesso problema, ma i loro rilassamenti continui non sono equivalenti.
  - Una formulazione Q_1 è migliore di Q_2 se $Q_1 \subset Q_2$, ovvero se il rilassamento continuo di Q_1 è più "stretto".
- **Formulazione ideale**:
  - La formulazione ideale di un problema di PLI è il **guscio convesso** (convex hull) dell'insieme delle soluzioni ammissibili.
  - Il guscio convesso è il più piccolo insieme convesso che contiene tutte le soluzioni ammissibili intere.

Algoritmi Generali per PLI

- **Metodi tradizionali**:
  - **Cutting Planes (Piani di Taglio)**: Aggiunge vincoli per "tagliare" le soluzioni frazionarie.
  - **Branch-and-Bound**: Divide il problema in sotto-problemi e utilizza limiti inferiori/superiori per escludere soluzioni non ottimali.
  - **Programmazione Dinamica**: Utile per problemi con struttura particolare.
- **Metodi avanzati**:
  - **Branch-and-Cut**: Combina Branch-and-Bound con Cutting Planes.
  - **Branch-and-Price/Column Generation**: Utilizzato per problemi con un numero elevato di variabili.

-Algoritmo Branch and Bound per la Programmazione Lineare Intera (PLI)

Introduzione all'Algoritmo Branch and Bound
L'algoritmo **Branch and Bound** (B&B) è una tecnica generale per risolvere problemi di ottimizzazione combinatoria, tra cui la Programmazione Lineare Intera (PLI). Si basa sulla suddivisione del problema originale in sottoproblemi più semplici, che vengono esplorati in modo sistematico.

Concetti Fondamentali

- **Problema originale** P_0:

$$z_0 = \min c^T x \quad \text{s.t.} \quad Ax \geq d, \quad x \geq 0, \quad x \in \mathbb{Z}$$

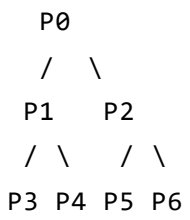
  - z_0: Valore ottimo della funzione obiettivo.
  - $F(P_0)$: Regione ammissibile del problema.
- **Sottoproblemi**:

- Il problema P_0 viene suddiviso in K sottoproblemi $P_1, P_2, \ldots, P_K$, ciascuno con una regione ammissibile $F(P_k)$.
- La suddivisione avviene in modo che:

$$F(P_0) = \bigcup_{k=1}^{K} F(P_k) \quad \text{e} \quad F(P_i) \cap F(P_j) = \emptyset \quad \text{per} \quad i \neq j$$

Rappresentazione ad Albero

- **Albero decisionale**:
  - I nodi rappresentano i sottoproblemi.
  - Gli archi rappresentano le relazioni di suddivisione (branching).
  - Esempio:



Passi dell'Algoritmo Branch and Bound

1. **Risolvi il rilassamento continuo**:
   - Per ogni sottoproblema P_k, risolvi il rilassamento continuo $C(P_k)$.
   - Ottieni una soluzione x_{Ck} e un valore z_{Ck}.
2. **Verifica l'ammissibilità**:
   - Se x_{Ck} è intera, allora è una soluzione ammissibile per P_k.
   - Se $z_{Ck} \geq z_{\text{Best}}$, il sottoproblema P_k può essere scartato (bounding).
3. **Branching**:
   - Se x_{Ck} non è intera, scegli una variabile frazionaria x_j e crea due nuovi sottoproblemi:
     - $P_{k1}: x_j \leq \lfloor x_{Ck,j} \rfloor$
     - $P_{k2}: x_j \geq \lfloor x_{Ck,j} \rfloor + 1$
4. **Esplorazione**:
   - Continua a esplorare i sottoproblemi attivi fino a quando non ci sono più nodi da esaminare.

Esempio di Applicazione

Problema PLI:

$$\max z = x_1 + x_2$$

soggetto a:

$$5x_1 + 3x_2 \leq 15, \quad 5x_1 - 3x_2 \geq 0, \quad x_2 \geq \frac{1}{2}, \quad x_1, x_2 \geq 0, \quad x_1, x_2 \in \mathbb{Z}$$

Passi:

1. **Risolvi il rilassamento continuo $C(P_0)$**:
   - Soluzione: $x_{C0} = \left(\frac{3}{2}, \frac{5}{2}\right)$, $z_{C0} = 4$.
2. **Branching su x_1**:
   - P_1: $x_1 \leq 1$
   - P_2: $x_1 \geq 2$
3. **Risolvi $C(P_1)$ e $C(P_2)$**:
   - P_1: $x_{C1} = (1, \frac{5}{3})$, $z_{C1} = \frac{8}{3}$.
   - P_2: $x_{C2} = (2, \frac{5}{3})$, $z_{C2} = \frac{11}{3}$.
4. **Branching su x_2 in P_2**:
   - P_3: $x_2 \leq 1$
   - P_4: $x_2 \geq 2$
5. **Risolvi $C(P_3)$ e $C(P_4)$**:
   - P_3: $x_{C3} = \left(\frac{12}{5}, 1\right)$, $z_{C3} = \frac{17}{5}$.
   - P_4: Impossibile.
6. **Branching su x_1 in P_3**:
   - P_5: $x_1 \leq 2$
   - P_6: $x_1 \geq 3$
7. **Risolvi $C(P_5)$**:
   - P_5: $x_{C5} = (2, 1)$, $z_{C5} = 3$ (soluzione intera).
8. **Terminazione**:
   - La soluzione ottima è $x^* = (2, 1)$ con $z^* = 3$.

Strategie di Esplorazione

- **Scelta del prossimo sottoproblema**:
  - Si preferisce esplorare il sottoproblema con il miglior limite superiore (per problemi di massimizzazione) o inferiore (per problemi di minimizzazione).
  - Esempio: Se $z_{C1} = \frac{8}{3}$ e $z_{C2} = \frac{11}{3}$, si esplora prima P_2.
- **Scelta della variabile per il branching**:
  - Si sceglie la variabile con la parte frazionaria più grande.

Terminologia

- **Nodo radice**: Il problema originale P_0.
- **Nodi foglia**: Sottoproblemi che non possono essere ulteriormente suddivisi.
- **Nodi attivi**: Sottoproblemi ancora da esplorare.
- **Bound**: Limite superiore/inferiore utilizzato per scartare sottoproblemi non promettenti.

Modelli di Programmazione Lineare Intera (PLI)

Introduzione ai Modelli PLI
La **Programmazione Lineare Intera (PLI)** è una tecnica di ottimizzazione in cui le variabili decisionali devono assumere valori **interi**. Questi modelli sono utilizzati in una vasta gamma di applicazioni, dalla pianificazione della produzione alla gestione delle risorse.

Modelli PLI con Esempi

Modello PLI per la Produzione di Mobili

Problema:

- **Furniturissimo** produce scrivanie, armadi e sedie di lusso.
- **Costi di setup settimanali**:
  - Scrivanie: 120 €
  - Armadi: 130 €
  - Sedie: 70 €
- **Requisiti di produzione**:
  - Ore di lavoro: 200 ore/settimana
  - Legno: 100 piedi quadrati/settimana
- **Obiettivo**: Massimizzare il profitto settimanale.

Dati:

Prodotto	Ore di Lavoro	Legno (piedi²)	Prezzo di Vendita (€)	Costi Variabili (€)
Scrivanie	16	10	330	82
Armadi	29	15	620	97
Sedie	10	3	150	28

Formulazione PLI:

- **Variabili decisionali**:

- x_1: Numero di scrivanie prodotte.
- x_2: Numero di armadi prodotti.
- x_3: Numero di sedie prodotte.

- **Funzione obiettivo**:

$$\max z = (330 - 82)x_1 + (620 - 97)x_2 + (150 - 28)x_3 - 120x_1 - 130x_2 - 70x_3$$

- **Vincoli**:

$$16x_1 + 29x_2 + 10x_3 \leq 200 \quad (\text{ore di lavoro})$$

$$10x_1 + 15x_2 + 3x_3 \leq 100 \quad (\text{legno})$$

$$x_1, x_2, x_3 \geq 0 \quad \text{e intere}$$

Modello PLI per Indagine di Mercato

Problema:

- **Obiettivo**: Minimizzare il costo delle telefonate per intervistare utenti in due fasce orarie (mattina e sera).
- **Costi**:
  - Mattina: 1 €/telefonata (almeno il 50% delle chiamate).
  - Sera: 1.5 €/telefonata.
- **Requisiti**:
  - Categorie di utenti da intervistare: A, B, C, D.

Formulazione PLI:

- **Variabili decisionali**:
  - x_1: Numero di telefonate al mattino.
  - x_2: Numero di telefonate alla sera.
- **Funzione obiettivo**:

$$\min z = x_1 + 1.5x_2$$

- **Vincoli**:

$$0.3x_1 + 0.3x_2 \geq 150 \quad (\text{Categoria A})$$

$$0.1x_1 + 0.2x_2 \geq 110 \quad \text{(Categoria B)}$$

$$x_1 \geq x_2 \quad \text{(Almeno il } 50/100 \text{ al mattino)}$$

$$x_1, x_2 \geq 0 \quad \text{e intere}$$

Modello PLI per il Noleggio di Macchinari

Problema:

- **Obiettivo**: Minimizzare il costo di noleggio di macchinari per 6 mesi.
- **Opzioni di noleggio**:
  - 1 mese: 400 €
  - 2 mesi: 700 €
  - 3 mesi: 900 €
- **Fabbisogno mensile**:

Mese	Gen	Feb	Mar	Apr	Mag	Giu
Fabbisogno	9	5	7	9	10	5

Formulazione PLI:

- **Variabili decisionali**:
  - $GE1, GE2, GE3$: Macchinari noleggiati a Gennaio per 1, 2, 3 mesi.
  - $FE1, FE2, FE3$: Macchinari noleggiati a Febbraio per 1, 2, 3 mesi.
  - ...
- **Funzione obiettivo**:

  $$\min z = 400(GE1 + FE1 + \dots) + 700(GE2 + FE2 + \dots) + 900(GE3 + FE3 + \dots)$$

- **Vincoli**:

$$GE1 + GE2 + GE3 \geq 9 \quad \text{(Gennaio)}$$

$$FE1 + FE2 + FE3 + GE2 + GE3 \geq 5 \quad \text{(Febbraio)}$$

$$\dots$$

$$GE1, GE2, \dots \geq 0 \quad \text{e intere}$$

Modello PLI per il Mix di Pubblicità

Problema:

- **Obiettivo**: Massimizzare il numero di utenti raggiunti con un budget di 150.000 €.
- **Canali pubblicitari**:
  - Giornali: 1.000 €/annuncio (max 30 annunci).
  - TV: 10.000 €/annuncio (max 15 annunci).
- **Utenti raggiunti**:
  - Giornali: 900 (1-10 annunci), 600 (11-20), 300 (21-30).
  - TV: 10.000 (1-5 annunci), 5.000 (6-10), 2.000 (11-15).

Formulazione PLI:

- **Variabili decisionali**:
  - $G1, G2, G3$: Annunci su giornali nelle 3 fasce.
  - $T1, T2, T3$: Annunci su TV nelle 3 fasce.
- **Funzione obiettivo**:

$$\max z = 900G1 + 600G2 + 300G3 + 10000T1 + 5000T2 + 2000T3$$

- **Vincoli**:

$$G1 + G2 + G3 + 10T1 + 10T2 + 10T3 \leq 150$$

$$G1, G2, G3 \leq 10, \quad T1, T2, T3 \leq 5$$

$$G1, G2, G3, T1, T2, T3 \geq 0 \quad \text{e intere}$$

Modello PLI per la Turnazione del Personale

Problema:

- **Obiettivo**: Minimizzare il numero di persone necessarie per coprire i turni settimanali.
- **Requisiti giornalieri**:

Giorno	Lun	Mar	Mer	Gio	Ven	Sab	Dom
Personale	22	18	13	14	15	18	25

Formulazione PLI:

- **Variabili decisionali**:
  - $x_1, x_2, \ldots, x_7$: Persone che iniziano il turno nei 7 giorni.
- **Funzione obiettivo**:

$$\min z = x_1 + x_2 + \cdots + x_7$$

- **Vincoli**:

$$x_1 + x_4 + x_5 + x_6 + x_7 \geq 22 \quad (\text{Lunedì})$$

$$x_1 + x_2 + x_5 + x_6 + x_7 \geq 18 \quad (\text{Martedì})$$

$$\ldots$$

$$x_1, x_2, \ldots, x_7 \geq 0 \quad \text{e intere}$$

Problema dello Zaino (Knapsack Problem - KP01)

- **Definizione**:
  - **n oggetti**, ciascuno con un **profitto** P_j e un **peso** W_j.
  - **1 zaino** con capacità massima K.
  - **Obiettivo**: Selezionare un sottoinsieme di oggetti che massimizzi il profitto totale, senza superare la capacità K.
- **Modello PLI**:

$$\max \sum_{j=1}^{n} P_j x_j$$

$$\sum_{j=1}^{n} W_j x_j \leq K$$

$$x_j \in \{0,1\} \quad (\text{dove } x_j = 1 \text{ se l'oggetto } j \text{ è selezionato, altrimenti } 0)$$

- **Esempio pratico**:
  - **Oggetti**: 8 articoli con valori e pesi specifici.
  - **Capacità zaino**: 21 lbs.
  - **Obiettivo**: Massimizzare il valore degli articoli selezionati senza superare i 21 lbs.

Problema dello Zaino Multiplo (Multi-Knapsack Problem - MKP01)

- **Definizione**:
  - **n oggetti**, ciascuno con profitto P_j e peso W_j.
  - **m contenitori**, ciascuno con capacità K_i.
  - **Vincolo aggiuntivo**: Ogni oggetto può essere assegnato a **al massimo un contenitore**.
  - **Obiettivo**: Massimizzare il profitto totale, rispettando i vincoli di capacità di ogni contenitore.
- **Modello PLI**:

$$\max \sum_{j=1}^{n} P_j \left(\sum_{i=1}^{m} x_{ij} \right)$$

$$\sum_{j=1}^{n} W_j x_{ij} \leq K_i \quad (\text{per ogni contenitore } i)$$

$$\sum_{i=1}^{m} x_{ij} \leq 1 \quad (\text{per ogni oggetto } j)$$

$$x_{ij} \in \{0,1\} \quad (\text{dove } x_{ij} = 1 \text{ se l'oggetto } j \text{ è assegnato al contenitore } i)$$

Problema del Bin Packing (1BP)

- **Definizione**:
  - **n oggetti**, ciascuno con peso W_j.
  - **n contenitori (bin)**, ciascuno con capacità K.

- **Obiettivo**: Impacchettare tutti gli oggetti nel **minor numero possibile di contenitori**, senza superare la capacità K in ogni contenitore.

- **Modello PLI**:

$$\min \sum_{i=1}^{n} y_i$$

$$\sum_{j=1}^{n} W_j x_{ij} \leq K y_i \quad (\text{per ogni contenitore } i)$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad (\text{per ogni oggetto } j)$$

$$y_i \in \{0,1\} \quad (\text{dove } y_i = 1 \text{ se il contenitore } i \text{ è utilizzato})$$

$$x_{ij} \in \{0,1\} \quad (\text{dove } x_{ij} = 1 \text{ se l'oggetto } j \text{ è assegnato al contenitore } i)$$

Problema di Assegnazione di Incarichi

- **Definizione**:
  - **n persone** e **n incarichi**.
  - **Costo** c_{ij}: Costo per assegnare l'incarico j alla persona i.
  - **Obiettivo**: Assegnare ogni persona a un incarico e ogni incarico a una persona, minimizzando il costo totale.
- **Modello PLI**:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad (\text{per ogni persona } i)$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad (\text{per ogni incarico } j)$$

$$x_{ij} \in \{0,1\} \quad (\text{dove } x_{ij} = 1 \text{ se la persona } i \text{ è assegnata all'incarico } j)$$

Problema di Sequenziamento di Lavorazioni

- **Definizione**:
  - **n lavorazioni**, ciascuna con tempo di processamento p_j.
  - **m macchine identiche**.
  - **Vincoli**:
    - Nessuna prelazione (una lavorazione non può essere interrotta).
    - Ogni macchina può eseguire una sola lavorazione alla volta.
  - **Obiettivo**: Assegnare le lavorazioni alle macchine in modo da minimizzare il **tempo totale di completamento (makespan)**.
- **Modello PL Misto**:

$$\min z$$

$$\sum_{j=1}^{n} p_j x_{ij} \leq z \quad (\text{per ogni macchina } i)$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad (\text{per ogni lavorazione } j)$$

$$x_{ij} \in \{0,1\} \quad (\text{dove } x_{ij} = 1 \text{ se la lavorazione } j \text{ è assegnata alla macchina } i)$$

$$z \geq 0 \quad (\text{variabile continua per il makespan})$$

Riepilogo delle Applicazioni

- **KP01**: Selezione ottimale di oggetti con vincoli di capacità.
- **MKP01**: Estensione a più contenitori con vincoli aggiuntivi.
- **1BP**: Ottimizzazione dell'uso dei contenitori.
- **Assegnazione di incarichi**: Ottimizzazione di costi/tempi in contesti di assegnazione.
- **Sequenziamento di lavorazioni**: Minimizzazione del tempo totale di completamento in contesti produttivi.

Constraint Programming (CP)

Concetti Fondamentali della Programmazione a Vincoli (CP)

La **Programmazione a Vincoli (CP)** è una tecnica di ottimizzazione che si basa su tre componenti principali:

1. **Variabili decisionali**:
   - Le variabili sono rappresentate da **domini**, che possono essere booleani, interi o numerici.
   - I domini finiti sono spesso rappresentati come intervalli (es. $x = [1, 5] = \{1, 2, 3, 4, 5\}$).
   - Se ci sono "buchi" nel dominio, questi vengono rappresentati esplicitamente (es. $x' = \{1, 3, 4, 10\}$).
2. **Vincoli**:
   - I vincoli collegano le variabili e possono essere lineari o non lineari.
   - Esistono vincoli predefiniti come `ALL_DIFFERENT`, che impone che un insieme di variabili assuma valori diversi.
   - Ogni vincolo ha un **propagatore**, che aggiorna i domini delle variabili in base alle implicazioni del vincolo.
3. **Funzione obiettivo**:
   - La funzione obiettivo può essere qualsiasi espressione matematica, inclusi operatori come `MAX`, `MIN`, `IF/THEN`, `POWER`.
   - È possibile accedere alle proprietà delle variabili di intervallo per costruire espressioni complesse.

Algoritmi di Risoluzione: Branch and Bound

L'algoritmo **Branch and Bound** è comunemente utilizzato nella CP per risolvere problemi di ottimizzazione. Ecco un esempio:

Problema:

$$\min 2X + Y$$

soggetto a:

$$2X + 3Y \geq 3, \quad X + Y \leq 4, \quad 2X - 3Y \geq 0, \quad X \leq 3, \quad X, Y \geq 0, \quad X, Y \in \mathbb{Z}$$

Passi:

1. **Rilassamento continuo**: Risolvi il problema senza vincoli di interezza.
2. **Branching**: Suddividi il problema in sottoproblemi aggiungendo vincoli (es. $X \leq 1$ e $X \geq 2$).
3. **Bound**: Utilizza i limiti inferiori/superiori per escludere sottoproblemi non promettenti.
4. **Esplorazione**: Continua fino a trovare la soluzione ottima intera.

Linguaggi di Modellazione

- **FlatZinc**: Linguaggio di modellazione di basso livello, standard per la CP.
- **MiniZinc**: Linguaggio di alto livello che permette di descrivere problemi in modo più intuitivo. MiniZinc traduce i modelli in FlatZinc per l'uso con vari solver.

Solver per la Programmazione a Vincoli

I solver CP sono strumenti software che risolvono problemi di ottimizzazione basati su vincoli. Ecco alcuni esempi:

- **IBM ILOG CP Optimizer**:
  - Interfacce: C/C++, Java, Python, OPL.
  - Focus su modelli interi e problemi di scheduling.
  - Include una libreria ricca di vincoli predefiniti.
  - Supporta l'ottimizzazione automatica e l'analisi dei conflitti.
- **GECODE**:
  - Solver open-source, gratuito e potente.
  - Supporta modelli con domini finiti e vincoli complessi.
- **CHOCO**:
  - Solver open-source scritto in Java.
  - Adatto per problemi di ottimizzazione combinatoria.
- **CHUFFED**:
  - Solver ibrido che combina tecniche SAT e CP.

Applicazioni di Successo della CP

La Programmazione a Vincoli è utilizzata in una vasta gamma di applicazioni, tra cui:

- **Scheduling**:

    - Pianificazione di turni di lavoro, produzione, trasporti.
    - Esempio: Assegnazione di risorse a task con vincoli temporali.
- **Timetabling**:
    - Creazione di orari per scuole, università, eventi.
    - Esempio: Assegnazione di aule e insegnanti a corsi.
- **Configurazione di prodotti**:
    - Personalizzazione di prodotti in base a vincoli specifici.
    - Esempio: Configurazione di automobili con opzioni compatibili.

Vantaggi e Svantaggi della CP

Vantaggi:

- **Facilità di modellazione**: I modelli CP sono facili da impostare e eseguire.
- **Vincoli complessi**: La CP permette di modellare vincoli non lineari e complessi.
- **Efficienza**: Trova soluzioni ammissibili rapidamente e identifica l'inammissibilità velocemente.
- **Soluzioni rapide**: Ideale per problemi con struttura combinatoria complessa.

Svantaggi:

- **Scalabilità**: Non è adatta per problemi di grandi dimensioni con poca struttura.
- **Qualità delle soluzioni**: Non garantisce direttamente la qualità della soluzione ottima.
- **Comportamento a scatola chiusa**: Alcuni solver possono comportarsi in modo imprevedibile.
- **Formulazione efficiente**: Richiede esperienza per formulare problemi in modo efficiente.

Risorse Utili

- **Libri e articoli**:
    - "IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems", Philippe Laborie, 2009.
    - "CP Optimizer Walkthrough", Paul Shaw, 2013.
    - "Modeling and Solving Scheduling Problems with CP Optimizer", Philippe Laborie, 2014.
    - "An Introduction to CP Optimizer", Philippe Laborie, 2016.
- **Siti web**:
    - **MiniZinc Challenge**: Competizione annuale di solver CP (www.minizinc.org/challenge.html).
    - **Global Constraint Catalogue**: Catalogo di vincoli globali (http://sofdem.github.io/gccat).
    - **MiniZinc Tool**: Strumento gratuito per modellazione e risoluzione di problemi CP (www.minizinc.org).

(VERIFICA DIAGRAMMI)

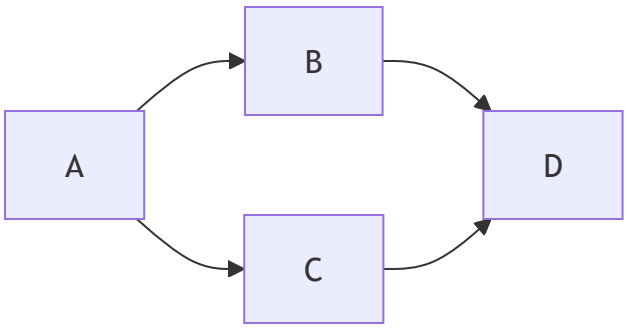
Teoria dei Grafi

1. Introduzione alla Teoria dei Grafi

La **Teoria dei Grafi** è una branca della matematica che studia le proprietà dei grafi, strutture composte da **vertici** (o nodi) e **lati** (o archi) che connettono coppie di vertici. I grafi sono utilizzati per modellare una vasta gamma di problemi reali, dalle reti di trasporto ai social network.

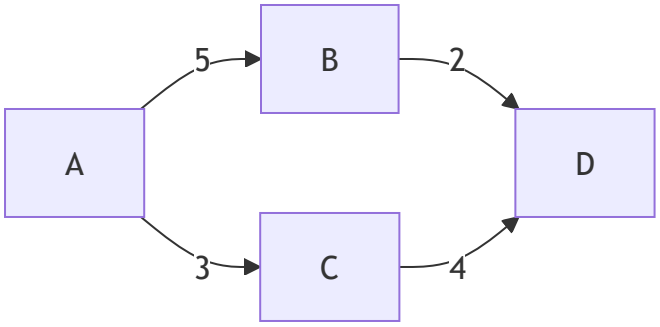
Grafi Non Orientati e Orientati

- **Grafi non orientati**:
  - Un grafo non orientato $G = (V, E)$ è definito da:
    - V: Insieme di vertici (nodi).
    - E: Insieme di lati, dove ogni lato $e = \{i, j\}$ connette due vertici i e j.
- **Grafi orientati**:
  - Un grafo orientato $G = (V, A)$ è definito da:
    - V: Insieme di vertici.
    - A: Insieme di archi, dove ogni arco $a = (i, j)$ connette il vertice i al vertice j.
  - Esempio:



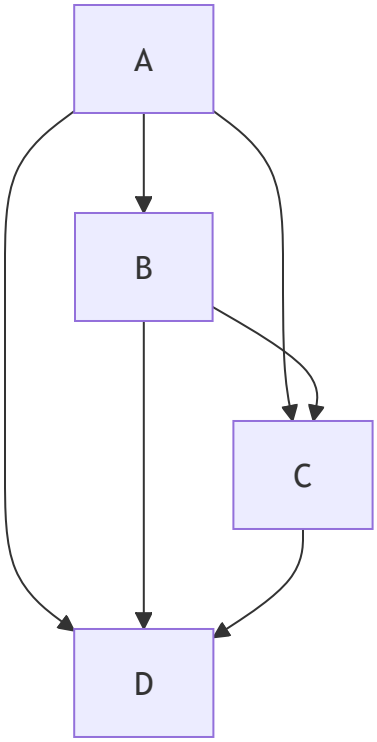
Grafi Pesati

- Un grafo è **pesato** se ai lati o agli archi è associato un peso (costo, distanza, ecc.).
- Esempio:



Grafi Multipli, Semplici e Completi

- **Grafi multipli**: Possono avere più lati tra la stessa coppia di vertici.
- **Grafi semplici**: Non hanno lati multipli né loop (lati che connettono un vertice a sé stesso).
- **Grafi completi**: Ogni coppia di vertici è connessa da un lato.
- Esempio di grafo completo:



Applicazioni dei Grafi

- **Cammini Euleriani**: Problema dei ponti di Königsberg.
- **Colorazione dei grafi**: Assegnazione di colori a regioni confinanti.
- **Problema della clique**: Trovare il sottografo completo più grande.

Taglio di un Grafo

In teoria dei grafi, **tagliare un grafo** si riferisce a una operazione che divide il grafo in due o più parti disgiunte, rimuovendo un insieme di archi o nodi. Questo concetto è fondamentale in diverse applicazioni, come l'ottimizzazione di reti, l'analisi di flussi, la segmentazione di immagini e molto altro.
Ecco una spiegazione dettagliata:
Un **taglio** è una partizione dei vertici (nodi) di un grafo in due sottoinsiemi disgiunti, solitamente indicati come S e T. Formalmente, dato un grafo $G = (V, E)$, un taglio è una partizione di V in due insiemi S e T tali che:

$$S \cup T = V \quad \text{e} \quad S \cap T = \emptyset$$

- **Archi del taglio**: Gli archi che collegano un nodo in S a un nodo in T sono detti **archi del taglio**.
- **Peso del taglio**: Se il grafo è pesato, il peso del taglio è la somma dei pesi degli archi del taglio.

Taglio in Grafi Non Orientati

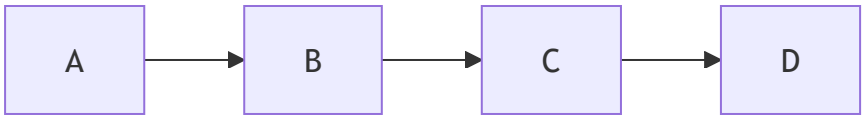
- Dato un sottoinsieme $S \subseteq V$, il **taglio** $\delta(S)$ è l'insieme dei lati che connettono S a $V \setminus S$.
  Se $S = \{A, B\}$, allora $\delta(S) = \{\{A, C\}, \{B, D\}\}$.

Taglio in Grafi Orientati

- In un grafo orientato, si distinguono:
  - **Archi uscenti** $\delta^+(S)$: Archi che partono da S e vanno a $V \setminus S$.
  - **Archi entranti** $\delta^-(S)$: Archi che partono da $V \setminus S$ e vanno a S.

Cammini

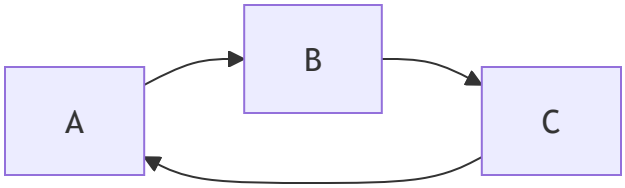
- Un **cammino** è una sequenza di vertici $v_1, v_2, \ldots, v_k$ tali che ogni coppia consecutiva è connessa da un lato o arco.
- Esempio:



Cammino: (A, B, C, D).

Circuiti e Cicli

- **Circuito**: Cammino in cui il primo e l'ultimo vertice coincidono.
- **Ciclo**: Versione non orientata di un circuito.
- Esempio di circuito:



Alberi

- Un **albero** è un grafo connesso e aciclico.
- Proprietà:
  - Ha $n - 1$ lati.
  - Se si rimuove un lato, il grafo diventa disconnesso.

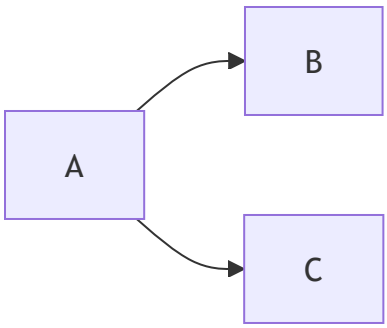
Rappresentazione dei Grafi

Matrice di Adiacenza

- Una matrice Q di dimensione $|V| \times |V|$ dove:

$$q_{ij} = \begin{cases} 1 & \text{se } \{i,j\} \in E \text{ (grafo non orientato)} \\ 1 & \text{se } (i,j) \in A \text{ (grafo orientato)} \\ 0 & \text{altrimenti} \end{cases}$$

- Esempio:



Matrice di adiacenza:

$$Q = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Matrice di Incidenza

- Una matrice D di dimensione $|V| \times |E|$ dove:

$$d_{ik} = \begin{cases} 1 & \text{se il lato } e_k \text{ è incidente su } i \\ 0 & \text{altrimenti} \end{cases}$$

Matrice di incidenza:

$$D = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Liste di Adiacenza

- Ogni vertice ha una lista dei vertici adiacenti.

Liste di adiacenza:

- A: $[B, C]$
- B: $[A]$
- C: $[A]$

Cammini Minimi (Shortest Paths)

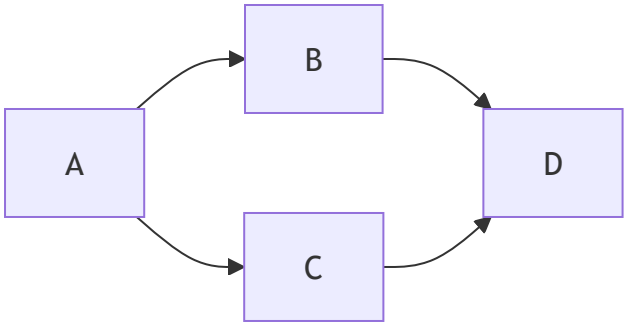
Il **problema del cammino minimo** consiste nel trovare il percorso più breve tra due nodi in una rete, dove la "distanza" può rappresentare costi, tempi, o altre metriche. Questo problema è fondamentale in applicazioni come la pianificazione di percorsi, il controllo di macchine e i sistemi di trasporto.

Concetti di Base

Reti e Notazione

- **Nodi (Vertici)**: Punti di intersezione o località.
- **Archi (Lati orientati)**: Collegamenti diretti tra nodi.
- **Pesi**: Valori associati agli archi (es. distanza, tempo).
- **Grado di un nodo**: Numero di vicini. Per reti orientate, si distingue tra **grado entrante** e **grado uscente**.

Esempio di rete orientata:



Problema del Cammino Minimo

Formulazione del Problema

- **Obiettivo**: Trovare il percorso più breve dal nodo A al nodo B.
- **Variabili decisionali**:
  - $x_{i,j} = 1$ se l'arco (i, j) è usato nel cammino minimo, 0 altrimenti.
- **Vincoli**:
  - **Flusso uscente dal nodo sorgente**:

$$\sum_{j \in \text{vicini uscenti di } A} x_{A,j} = 1$$

  - **Flusso entrante nel nodo sorgente**:

$$\sum_{j \in \text{vicini entranti di } A} x_{j,A} = 0$$

- **Conservazione del flusso nei nodi intermedi**:

$$\sum_{i \in \text{vicini entranti di } v} x_{i,v} = \sum_{j \in \text{vicini uscenti di } v} x_{v,j}$$

- **Flusso entrante nel nodo destinazione**:

$$\sum_{i \in \text{vicini entranti di } B} x_{i,B} = 1$$

- **Funzione obiettivo**:

$$\min \sum_{(i,j) \in A} w_{i,j} x_{i,j}$$

Algoritmo di Dijkstra (1956)
L'algoritmo di Dijkstra trova il cammino minimo da un nodo sorgente a tutti gli altri nodi in una rete con pesi non negativi.

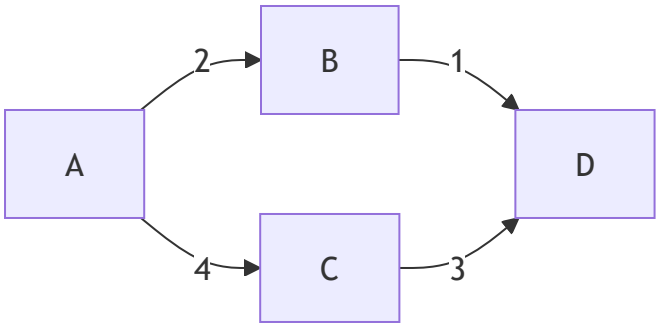
Passi:

1. **Inizializzazione**:
   - Imposta $d(i) = \infty$ per ogni nodo i.
   - Imposta $d(A) = 0$ per il nodo sorgente A.
   - Segna tutti i nodi come non visitati.
2. **Selezione del nodo**:
   - Scegli il nodo non visitato i con la distanza minima $d(i)$.
3. **Aggiornamento delle distanze**:
   - Per ogni nodo j raggiungibile da i:

$$d(j) = \min(d(j), d(i) + w_{i,j})$$

     - Se $d(j)$ viene aggiornato, imposta i come predecessore di j: $\text{PRED}(j) = i$.
4. **Marcatura**:
   - Segna i come visitato.
5. **Terminazione**:
   - Ripeti i passi 2-4 fino a quando tutti i nodi sono visitati.
6. **Risultato**:
   - Il cammino minimo è codificato nel vettore PRED.

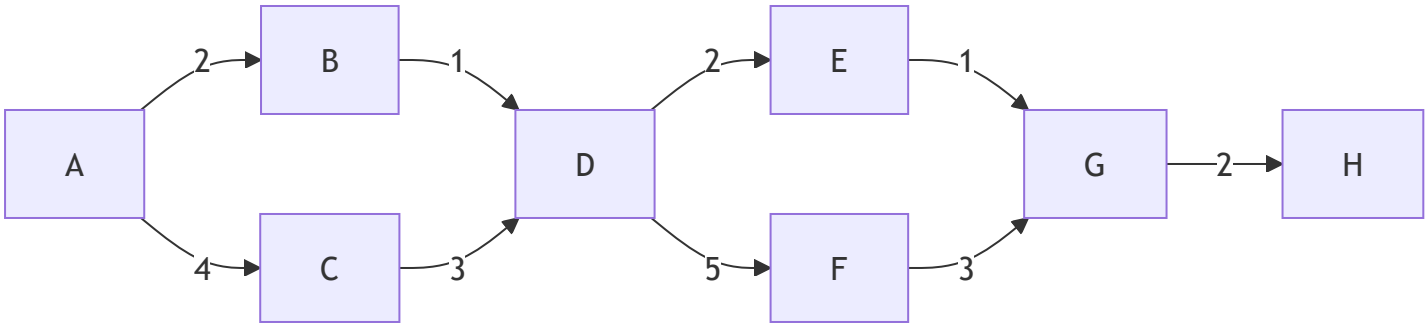
Esempio:



- **Cammino minimo da A a D**: $A \to B \to D$ con distanza 3.

Esempi di Applicazione

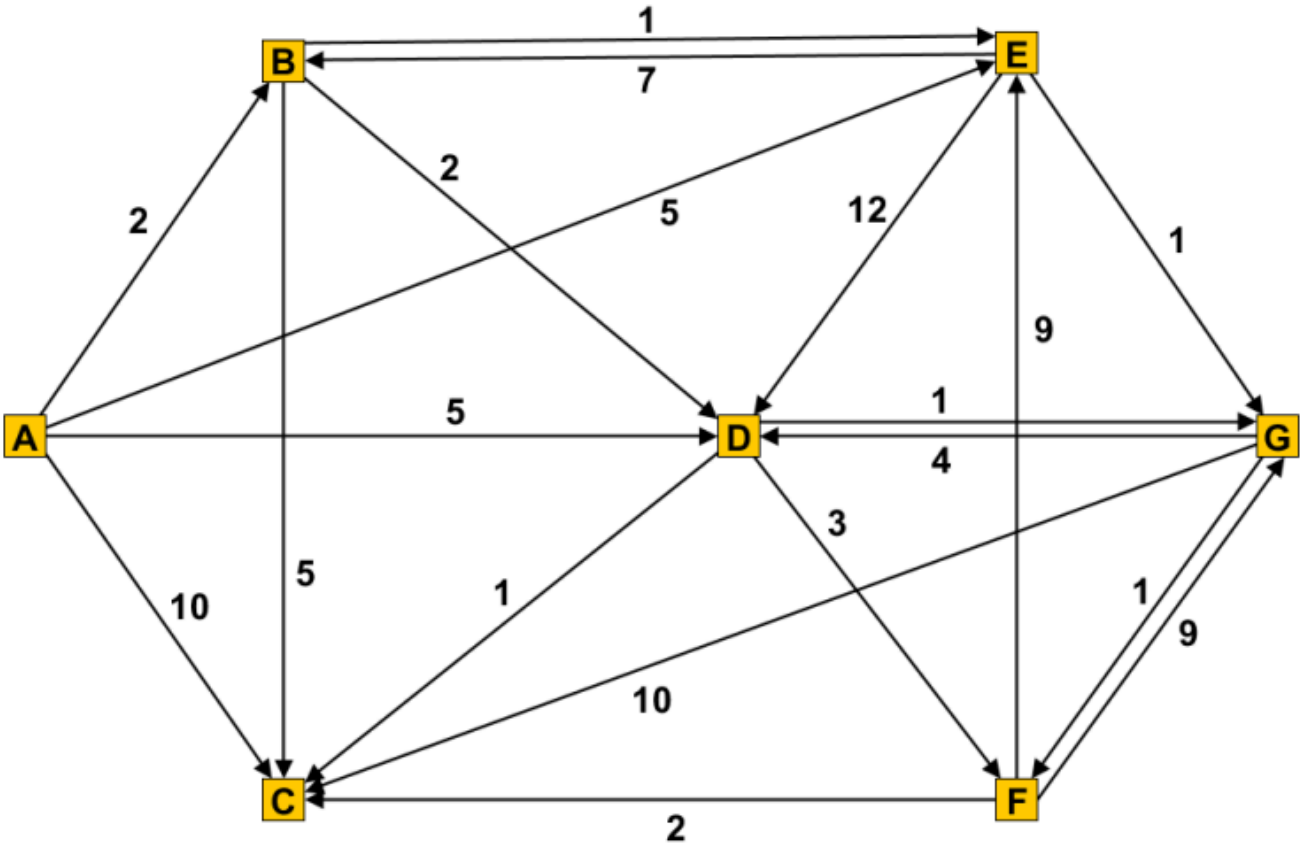
Esempio 1: Cammino Minimo da A a H



- **Cammino minimo**: $A \to B \to D \to E \to G \to H$ con distanza 8.

Esercizi

Esercizio 1: Applicazione dell'Algoritmo di Dijkstra



Costruisci un modello di Programmazione Lineare Intera (PLI) per trovare i cammini minimi:

- **c) Da A a G.**
- **d) Da C a E.**

Passo 1: Definire le Variabili Decisionali

- Per ogni arco (i, j) nel grafo, definiamo una variabile binaria:

$$x_{ij} = \begin{cases} 1 & \text{se l'arco } (i,j) \text{ è incluso nel cammino} \\ 0 & \text{altrimenti} \end{cases}$$

Passo 2: Funzione Obiettivo

Minimizzare il **costo totale del cammino**, dove c_{ij} è il costo dell'arco (i, j):

$$\text{Min } Z = \sum_{(i,j) \in E} c_{ij} \cdot x_{ij}$$

Passo 3: Vincoli

1. **Vincoli di Conservazione del Flusso**:
   - **Nodo sorgente** (es. A per A→G o C per C→E):

$$\sum_{(s,j)\in E} x_{sj} - \sum_{(k,s)\in E} x_{ks} = 1$$

- **Nodo destinazione** (es. G per A→G o E per C→E):

$$\sum_{(k,d)\in E} x_{kd} - \sum_{(d,j)\in E} x_{dj} = 1$$

- **Nodi intermedi** (tutti gli altri nodi):

$$\sum_{(i,k)\in E} x_{ik} = \sum_{(k,j)\in E} x_{kj} \quad \forall k \neq s, d$$

Cosa significa "conservazione del flusso"?
Immagina di dover inviare **1 unità di acqua** dal nodo A (sorgente) al nodo G (destinazione). I nodi intermedi (B, C, D, E) sono come **serbatoi**: l'acqua che entra deve uscire tutta, senza accumularsi.

- **Se entra 1 unità in un nodo, deve uscirne 1**.
- **Se non entra acqua, non può uscirne**.

Esempio Pratico: Nodo B
Supponiamo che il nodo B abbia:

- **Arco entrante**: $A \to B$.
- **Archi uscenti**: $B \to C$, $B \to E$.

Vincolo di conservazione per B:

$$x_{AB} = x_{BC} + x_{BE}$$

- **Se $x_{AB} = 1$** (l'acqua passa da A a B):
  Deve uscire 1 unità da B, quindi:

$$x_{BC} + x_{BE} = 1 \quad (\text{o } B \to C \text{ o } B \to E \text{ è attivo}).$$

- **Se $x_{AB} = 0$** (nessun flusso entra in B):

$$x_{BC} + x_{BE} = 0 \quad (\text{nessun arco uscente da } B \text{ è attivo}).$$

Alberi Ricoprenti (Spanning Trees)

Un **albero ricoprente** (spanning tree) di un grafo è un sottografo che include tutti i vertici del grafo ed è connesso senza cicli. Il **problema dell'albero ricoprente minimo** (MST, Minimum Spanning Tree) consiste nel trovare un albero ricoprente con il costo totale minimo.

Definizione di Albero

- Un **albero** è un grafo connesso e aciclico.
- Ogni coppia di nodi è connessa da un unico cammino.
- Un albero con n nodi ha esattamente $n - 1$ lati.

Problema dell'Albero Ricoprente Minimo (MST)

Formulazione del Problema

- **Obiettivo**: Trovare un albero ricoprente che connetta tutti i nodi con il costo totale minimo.
- **Variabili decisionali**:
  - $x_{i,j} = 1$ se il lato $\{i, j\}$ è incluso nell'albero, 0 altrimenti.
- **Vincoli**:
  - **Numero di lati**:

$$\sum_{\{i,j\} \in E} x_{i,j} = n - 1$$

  - **Assenza di cicli**:

$$\sum_{\{i,j\} \in E(S)} x_{i,j} \leq |S| - 1 \quad \text{per ogni sottoinsieme } S \subset N$$

- **Funzione obiettivo**:

$$\min \sum_{\{i,j\} \in E} w_{i,j} x_{i,j}$$

Algoritmo di Prim (1930)
L'algoritmo di Prim è un metodo efficiente per trovare l'albero ricoprente minimo in un grafo connesso e pesato.

Passi:

1. **Inizializzazione**:
   - Seleziona un nodo arbitrario come punto di partenza.
   - Inizializza un albero parziale con questo nodo.
2. **Aggiunta di lati**:
   - Trova il lato di costo minimo che connette l'albero parziale a un nodo non ancora incluso.
   - Aggiungi questo lato all'albero parziale.
3. **Terminazione**:
   - Ripeti il passo 2 fino a quando tutti i nodi sono inclusi nell'albero.

Modello PLI per l'MST

1. Variabili Decisionali

Per ogni arco $\{i, j\} \in E$, definiamo una **variabile binaria**:

$$x_{ij} = \begin{cases} 1 & \text{se l'arco } \{i,j\} \text{ è incluso nell'albero} \\ 0 & \text{altrimenti} \end{cases}$$

2. Funzione Obiettivo

Minimizzare il **costo totale** dell'albero:

$$\text{Min } Z = \sum_{\{i,j\} \in E} c_{ij} \cdot x_{ij}$$

3. Vincoli

1. **Numero di archi**:
   Un albero ricoprente deve avere esattamente $|N| - 1$ archi (dove $|N|$ è il numero di nodi).
   Vincolo:
   $$\sum_{\{i,j\} \in E} x_{ij} = |N| - 1$$
2. **Assenza di cicli**:
   Per evitare cicli, dobbiamo garantire che per ogni **sottoinsieme di nodi** $S \subset N$, il numero di archi selezionati all'interno di S sia al massimo $|S| - 1$.
   Vincolo:
   $$\sum_{\{i,j\} \in E(S)} x_{ij} \leq |S| - 1 \quad \forall S \subset N, S \neq \emptyset$$
   - $E(S)$ è l'insieme degli archi i cui estremi sono entrambi in S.
   - Questo vincolo garantisce che non si formino cicli all'interno di nessun sottoinsieme di nodi.

Esempio Pratico

Consideriamo un grafo con 4 nodi (A, B, C, D) e i seguenti archi con i relativi costi:

- $\{A, B\}$, costo 1
- $\{A, C\}$, costo 4
- $\{B, C\}$, costo 2
- $\{B, D\}$, costo 5
- $\{C, D\}$, costo 3

Variabili Decisionali:

$$x_{AB}, x_{AC}, x_{BC}, x_{BD}, x_{CD}$$

Funzione Obiettivo:

$$\text{Min } Z = 1x_{AB} + 4x_{AC} + 2x_{BC} + 5x_{BD} + 3x_{CD}$$

Vincoli:

1. **Numero di archi**:

$$x_{AB} + x_{AC} + x_{BC} + x_{BD} + x_{CD} = 3 \quad (\text{poiché } |N| = 4)$$

2. **Assenza di cicli**:
   - Per $S = \{A, B, C\}$:

$$x_{AB} + x_{AC} + x_{BC} \leq 2$$

   - Per $S = \{B, C, D\}$:

$$x_{BC} + x_{BD} + x_{CD} \leq 2$$

   - Per $S = \{A, B, D\}$:

$$x_{AB} + x_{BD} \leq 2$$

   - E così via per tutti i sottoinsiemi $S \subset N$.

Osservazioni Importanti

1. **Numero di vincoli**:
   - Il vincolo di assenza di cicli richiede di considerare **tutti i sottoinsiemi** $S \subset N$.
   - Questo porta a un numero esponenziale di vincoli, rendendo il modello complesso da risolvere per grafi grandi.
2. **Alternative pratiche**:
   - In pratica, si utilizzano algoritmi specifici per l'MST (es. **Kruskal** o **Prim**), che sono più efficienti della PLI.
   - La PLI è utile per estensioni del problema (es. MST con vincoli aggiuntivi).

Riepilogo del Modello PLI

- **Variabili**: $x_{ij} \in \{0, 1\}$ per ogni arco $\{i, j\} \in E$.
- **Funzione obiettivo**: Minimizzare il costo totale.
- **Vincoli**:

i. Numero di archi $= |N| - 1$.
ii. Assenza di cicli per ogni sottoinsieme $S \subset N$.

Problema del Commesso Viaggiatore (TSP)

Il **Problema del Commesso Viaggiatore (TSP)** consiste nel trovare il percorso più breve che un commesso viaggiatore può seguire per visitare un insieme di città, passando per ciascuna città esattamente una volta e tornando al punto di partenza. Questo problema è **NP-hard**, il che significa che è estremamente difficile da risolvere in modo ottimale per grandi istanze.

Formulazione del Problema

Modello di Programmazione Lineare Intera (PLI)

- **Variabili decisionali**:
  - $x_{i,j} = 1$ se l'arco (i, j) è incluso nel tour, 0 altrimenti.
- **Vincoli**:
  - **Un solo arco uscente da ogni nodo**:

$$\sum_{j \in N} x_{i,j} = 1 \quad \text{per ogni nodo } i \in N$$

  - **Un solo arco entrante in ogni nodo**:

$$\sum_{i \in N} x_{i,j} = 1 \quad \text{per ogni nodo } j \in N$$

  - **Eliminazione dei sottotour**:
    - **Vincoli SEC (Subtour Elimination Constraints)**:

$$\sum_{(i,j) \in A(S)} x_{i,j} \leq |S| - 1 \quad \text{per ogni sottoinsieme } S \subset N$$

    - **Vincoli MTZ (Miller-Tucker-Zemlin)**:

$$u_i - u_j + 1 \leq |N| \cdot (1 - x_{i,j}) \quad \text{per ogni } i, j \in N, i, j \neq 1$$

$$u_1 = 1$$

$$u_i \geq 2 \quad \text{per ogni } i \in N \setminus \{1\}$$

a. **Variabili ausiliarie**:
   - u_i: Variabile intera che rappresenta l'ordine di visita del nodo i.
   - $x_{i,j}$: Variabile binaria che indica se l'arco (i, j) è incluso nel percorso/albero.

b. **Vincoli MTZ**:

$$u_i - u_j + 1 \leq |N| \cdot (1 - x_{i,j}) \quad \text{per ogni } i, j \in N, i, j \neq 1$$

   - $|N|$: Numero totale di nodi nel grafo.
   - Questo vincolo garantisce che, se $x_{i,j} = 1$ (l'arco (i, j) è attivo), allora $u_i < u_j$, ovvero il nodo i viene visitato prima del nodo j.

c. **Vincolo sul nodo iniziale**:

$$u_1 = 1$$

   - Il nodo 1 è il **nodo iniziale** e ha ordine di visita 1.

d. **Vincoli sugli ordini di visita**:

$$u_i \geq 2 \quad \text{per ogni } i \in N \setminus \{1\}$$

   - Tutti gli altri nodi hanno un ordine di visita maggiore o uguale a 2.

Come Funzionano i Vincoli MTZ?

- **Se** $x_{i,j} = 1$:
  Il vincolo diventa:

$$u_i - u_j + 1 \leq 0 \quad \Rightarrow \quad u_i < u_j$$

  Questo impone che il nodo i sia visitato prima del nodo j.
- **Se** $x_{i,j} = 0$:
  Il vincolo diventa:

$$u_i - u_j + 1 \leq |N|$$

  Questo è sempre soddisfatto, poiché u_i e u_j sono limitati superiormente da $|N|$.

- **Funzione obiettivo**:

$$\min \sum_{(i,j) \in A} c_{i,j} x_{i,j}$$

Algoritmi Euristici

Euristica del Vicino più Vicino (Nearest Neighbor, NN)
L'euristica del Vicino più Vicino è un metodo semplice per trovare una soluzione approssimata al TSP.

Passi:

1. **Inizializzazione**:
   - Inizia da un nodo arbitrario.
   - Inizializza il tour con questo nodo.
2. **Selezione del prossimo nodo**:
   - Trova il nodo più vicino al nodo corrente che non è ancora nel tour.
   - Aggiungi questo nodo al tour.
3. **Terminazione**:
   - Ripeti il passo 2 fino a quando tutti i nodi sono inclusi nel tour.
   - Chiudi il tour tornando al nodo di partenza.

Ricerca Locale: 2-Opt
L'algoritmo 2-Opt è una tecnica di ricerca locale che migliora un tour esistente scambiando coppie di archi.

Passi:

1. **Inizializzazione**:
   - Parti da un tour iniziale.
2. **Scambio di archi**:
   - Trova due archi (i, j) e (k, l) nel tour.
   - Rimuovi questi archi e aggiungi (i, k) e (j, l).
   - Inverti il percorso tra j e k.
3. **Terminazione**:
   - Ripeti il passo 2 fino a quando non è possibile migliorare ulteriormente il tour.

Per valutare le prestazioni degli algoritmi euristici come l'**Euristica del Vicino più Vicino (NN)** e la **Ricerca Locale 2-Opt**, puoi utilizzare il **Minimum Spanning Tree (MST)** come strumento di confronto e analisi. Ecco come puoi farlo:

Utilizzo dell'MST come benchmark
L'MST (Minimum Spanning Tree) è un albero che connette tutti i nodi di un grafo con il costo minimo totale. Nel contesto del TSP (Traveling Salesman Problem), l'MST può essere utilizzato come limite inferiore per il costo ottimale del tour. Questo perché il costo dell'MST è sempre inferiore o uguale al costo del tour ottimale del TSP.

Valutazione dell'Euristica del Vicino più Vicino (NN)

- **Passi**:
  i. Applica l'euristica NN per ottenere un tour.
  ii. Calcola il costo del tour NN.
  iii. Confronta il costo del tour NN con il costo dell'MST.
  iv. Calcola il **rapporto di approssimazione**:

$$\text{Rapporto} = \frac{\text{Costo del tour NN}}{\text{Costo dell'MST}}$$

  v. Ripeti il processo su diverse istanze del problema per valutare la consistenza dell'euristica.

Valutazione della Ricerca Locale 2-Opt

- **Passi**:
  i. Applica l'euristica NN per ottenere un tour iniziale.
  ii. Migliora il tour utilizzando l'algoritmo 2-Opt.
  iii. Calcola il costo del tour migliorato.
  iv. Confronta il costo del tour 2-Opt con il costo dell'MST.
  v. Calcola il **rapporto di approssimazione**:

$$\text{Rapporto} = \frac{\text{Costo del tour 2-Opt}}{\text{Costo dell'MST}}$$

  vi. Ripeti il processo su diverse istanze per valutare l'efficacia del 2-Opt.
- **Limiti dell'MST**:
  - Ricorda che l'MST fornisce solo un limite inferiore. Il costo ottimale del TSP potrebbe essere significativamente più alto, specialmente in grafi con strutture complesse.

Algoritmo di Christofides

L'algoritmo di Christofides è un metodo euristico per risolvere il **problema del commesso viaggiatore (TSP)** su grafi metrici, ovvero grafi in cui vale la **disuguaglianza triangolare** (il costo diretto tra due nodi è sempre minore o uguale al costo di un percorso che passa attraverso un nodo intermedio). Proposto da Nicos Christofides nel 1976, questo algoritmo garantisce un **rapporto di approssimazione di 1.5**, il che significa che il costo del tour trovato è al massimo 1.5 volte il costo del tour ottimale. Ecco i passaggi dettagliati:

1. Calcolo del Minimum Spanning Tree (MST)

- **Obiettivo**: Trovare un albero che connette tutti i nodi del grafo con il costo minimo totale.
- **Metodo**: Utilizza algoritmi come quello di Kruskal o Prim per costruire l'MST.
- **Significato**: L'MST rappresenta una struttura di base che collega tutti i nodi senza cicli e con il costo minimo possibile.

2. Identificazione dei nodi di grado dispari

- **Obiettivo**: Trovare tutti i nodi nell'MST che hanno un numero dispari di archi incidenti (grado dispari).
- **Proprietà**: In qualsiasi grafo, il numero di nodi di grado dispari è sempre pari.
- **Significato**: Questi nodi rappresentano i punti in cui il percorso non è bilanciato e devono essere "corretti" per creare un grafo euleriano.

3. Matching perfetto di costo minimo

- **Obiettivo**: Trovare un **matching perfetto** (accoppiamento) tra i nodi di grado dispari, in modo che ogni nodo sia connesso a un altro nodo e il costo totale degli archi del matching sia minimo.
- **Metodo**: Utilizza algoritmi come quello di Edmonds per trovare il matching perfetto di costo minimo.
- **Significato**: Il matching aggiunge archi al grafo per bilanciare i nodi di grado dispari, trasformando l'MST in un grafo euleriano.

4. Creazione del grafo euleriano

- **Obiettivo**: Unire l'MST e il matching perfetto per formare un grafo in cui tutti i nodi hanno grado pari (grafo euleriano).
- **Proprietà**: In un grafo euleriano, è possibile trovare un ciclo che passa attraverso ogni arco esattamente una volta.
- **Significato**: Questo grafo è la base per costruire un percorso che visita tutti i nodi.

5. Costruzione del tour hamiltoniano

- **Obiettivo**: Trasformare il percorso euleriano in un **tour hamiltoniano** (un ciclo che visita ogni nodo esattamente una volta).
- **Metodo**:
  i. Trova un ciclo euleriano nel grafo euleriano.
  ii. Percorri il ciclo euleriano, "saltando" i nodi già visitati per evitare ripetizioni.
- **Significato**: Questo passaggio garantisce che il tour sia valido per il TSP, rispettando il vincolo di visitare ogni nodo una sola volta.

6. Garanzia di approssimazione

- **Teorema**: L'algoritmo di Christofides garantisce che il costo del tour trovato sia al massimo **1.5 volte** il costo del tour ottimale.
- **Dimostrazione**:
  - Il costo dell'MST è inferiore o uguale al costo del tour ottimale.
  - Il costo del matching perfetto è al massimo la metà del costo del tour ottimale.

- Quindi, il costo totale del tour è al massimo $\text{MST} + \text{Matching} \leq \text{OPT} + 0.5 \cdot \text{OPT} = 1.5 \cdot \text{OPT}$.

7. Vantaggi e limiti

- **Vantaggi**:
  - Garantisce una soluzione di alta qualità con un rapporto di approssimazione noto.
  - Efficace per grafi metrici che rispettano la disuguaglianza triangolare.
- **Limiti**:
  - Non è applicabile a grafi non metrici.
  - Richiede il calcolo di un matching perfetto, che può essere computazionalmente costoso per grafi molto grandi.

Algoritmo di Simulated Annealing per il TSP

Il **Simulated Annealing** (SA) è una meta-euristica ispirata al processo di ricottura in metallurgia, utilizzata per risolvere problemi di ottimizzazione come il TSP. A differenza degli algoritmi deterministici, il SA esplora lo spazio delle soluzioni in modo probabilistico, permettendo di evitare minimi locali accettando occasionalmente soluzioni peggiori. Ecco i passaggi dettagliati applicati al TSP:

1. Inizializzazione

- **Soluzione iniziale**: Genera una soluzione iniziale, ad esempio utilizzando l'euristica del Vicino più Vicino (NN) o una permutazione casuale dei nodi.
- **Temperatura iniziale (T_0)**: Imposta una temperatura iniziale alta, che determina la probabilità di accettare soluzioni peggiori.
- **Parametri**: Definisci il tasso di raffreddamento (α), il numero di iterazioni per ogni temperatura e il criterio di arresto (es. temperatura minima T_{min} o numero massimo di iterazioni).

2. Ciclo principale

Ripeti i seguenti passi finché non viene raggiunto il criterio di arresto:

a. Generazione di una soluzione vicina

- **Mossa**: Modifica la soluzione corrente applicando una piccola perturbazione, ad esempio:
  - **2-Opt swap**: Seleziona due archi del tour e scambiali per creare un nuovo percorso.
  - **Inversione di un sotto-tour**: Inverti l'ordine di visita di un sottoinsieme di nodi.
- **Calcolo del costo**: Calcola il costo della nuova soluzione (C_{new}) e confrontalo con il costo della soluzione corrente ($C_{current}$).

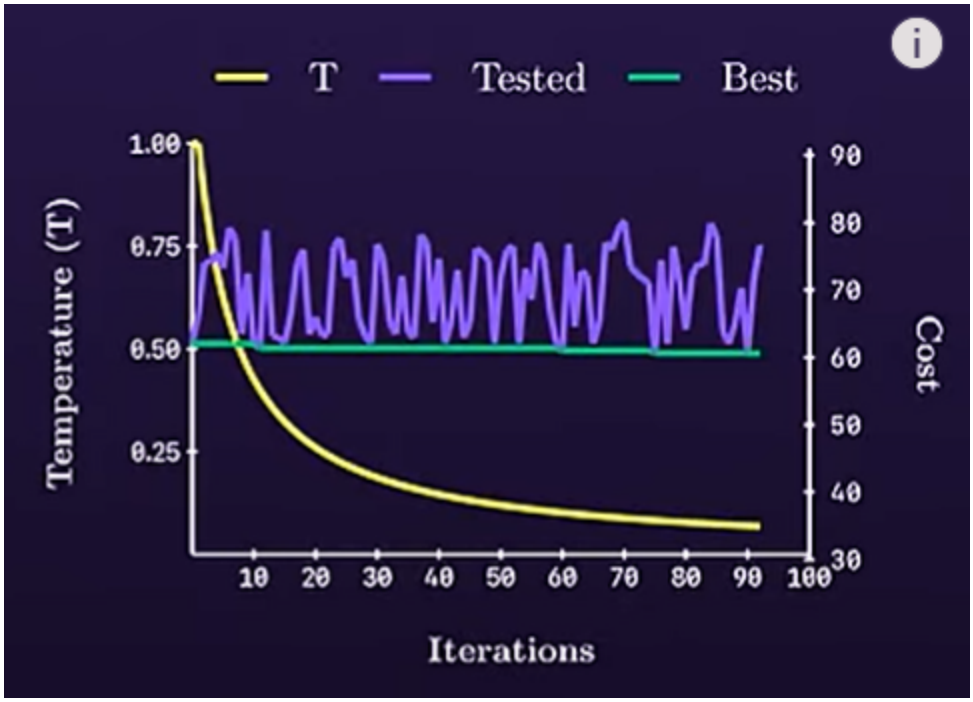
b. Accettazione della nuova soluzione

- **Se $C_{new} < C_{current}$**: Accetta sempre la nuova soluzione (miglioramento).

- **Se** $C_{new} \geq C_{current}$: Accetta la nuova soluzione con probabilità:

$$P = e^{-\frac{(C_{new} - C_{current})}{T}}$$

dove T è la temperatura corrente. Questa probabilità diminuisce con l'aumentare della differenza di costo e con la riduzione della temperatura.



c. Aggiornamento della temperatura

- Riduci la temperatura secondo il tasso di raffreddamento:

$$T = \alpha \cdot T$$

dove α è tipicamente compreso tra 0.95 e 0.99.

3. Terminazione

- **Criterio di arresto**: L'algoritmo termina quando la temperatura raggiunge un valore minimo (T_{min}) o dopo un numero massimo di iterazioni.
- **Soluzione finale**: Restituisci la migliore soluzione trovata durante l'esecuzione.

Parametri chiave e tuning

- **Temperatura iniziale (T_0)**: Deve essere abbastanza alta da permettere l'esplorazione iniziale di soluzioni peggiori.

- **Tasso di raffreddamento (α)**: Controlla la velocità con cui la temperatura diminuisce. Valori più alti permettono un'esplorazione più lunga.
- **Numero di iterazioni per temperatura**: Determina quante mosse vengono eseguite a ogni livello di temperatura.

Vantaggi e limiti

- **Vantaggi**:
  - Può evitare minimi locali grazie all'accettazione di soluzioni peggiori.
  - Flessibile e adattabile a diversi problemi di ottimizzazione.
- **Limiti**:
  - Richiede un tuning accurato dei parametri per ottenere buoni risultati.
  - Computazionalmente costoso per istanze molto grandi.

Problema del Commesso Viaggiatore Asimmetrico (ATSP)

Il **Problema del Commesso Viaggiatore Asimmetrico (ATSP)** è una variante del classico TSP in cui i costi di viaggio tra due nodi possono essere diversi a seconda della direzione. Questo problema è definito su una rete diretta, dove il commesso viaggiatore deve visitare ogni nodo esattamente una volta e tornare al punto di partenza, minimizzando il costo totale del percorso. Anche l'ATSP è un problema **NP-hard**, rendendo difficile la risoluzione ottimale per istanze di grandi dimensioni.

Formulazione del Problema

Modello di Programmazione Lineare Intera (PLI)

- **Variabili decisionali**:
  - $x_{i,j} = 1$ se l'arco (i, j) è incluso nel tour, 0 altrimenti.
- **Vincoli**:
  - **Un solo arco uscente da ogni nodo**:

$$\sum_{j \in V} x_{i,j} = 1 \quad \text{per ogni nodo } i \in V$$

  - **Un solo arco entrante in ogni nodo**:

$$\sum_{i \in V} x_{i,j} = 1 \quad \text{per ogni nodo } j \in V$$

  - **Eliminazione dei sottotour**:

- **Vincoli SEC (Subtour Elimination Constraints)**:

$$\sum_{i \in S} \sum_{j \in S} x_{i,j} \leq |S| - 1 \quad \text{per ogni sottoinsieme } S \subset V, |S| \geq 2$$

- **Vincoli MTZ (Miller-Tucker-Zemlin)**:

$$u_i - u_j + n \cdot x_{i,j} \leq n - 1 \quad \text{per ogni } i, j \in V, i \neq j$$

$$u_1 = 1$$

$$2 \leq u_i \leq n \quad \text{per ogni } i \in V \setminus \{1\}$$

dove u_i rappresenta la posizione del nodo i nel tour.

- **Funzione obiettivo**:

$$\min \sum_{(i,j) \in A} c_{i,j} x_{i,j}$$

Miglioramento dei Vincoli MTZ

Un miglioramento dei vincoli MTZ è stato proposto da Sawik (2016) per l'ATSP, introducendo vincoli aggiuntivi che tengono conto dell'ordinamento dei nodi di confine (i nodi immediatamente successivi e precedenti al deposito) e di tutti i nodi intermedi nel tour.

Vincoli di Confine:

1. **Primo nodo del tour**:
   Se il nodo j è il primo nodo visitato dopo il deposito, allora $u_j = u_1 + 1$. Questo può essere modellato come:

$$u_j - u_1 + (n-2) \cdot x_{1,j} \leq n - 1 \quad \text{per ogni } j \in V, j > 1$$

2. **Ultimo nodo del tour**:
   Se il nodo i è l'ultimo nodo visitato prima di tornare al deposito, allora $u_i = u_1 + n - 1$. Questo può essere modellato come:

$$u_1 - u_i + (n-1) \cdot x_{i,1} \leq 0 \quad \text{per ogni } i \in V, i > 1$$

Vincoli Intermedi:
Per ogni coppia di nodi successivi (i, j) nel tour, se $x_{i,j} = 1$, allora $u_j = u_i + 1$. Questo può essere modellato come:

$$1 - n + n \cdot x_{i,j} \leq u_j - u_i \leq n - 1 - (n - 2) \cdot x_{i,j} \quad \text{per ogni } i, j \in V, i \neq j, i > 1, j > 1$$

Problema della Cricca Massima (Maximum Clique Problem)

Il **Problema della Cricca Massima** consiste nel trovare, in un grafo non orientato $G = (V, E)$, un sottoinsieme di nodi $C \subseteq V$ tale che ogni coppia di nodi in C sia connessa da un arco, e la cardinalità di C sia massima. Questo problema è **NP-hard**, il che lo rende estremamente difficile da risolvere in modo ottimale per istanze di grandi dimensioni.

Applicazioni

- **Biologia**: Identificazione di gruppi di geni correlati.
- **Analisi delle reti sociali**: Trovare gruppi di individui fortemente connessi.
- **Telecomunicazioni**: Ottimizzazione delle reti di comunicazione.
- **Informatica**: Risoluzione di problemi di ottimizzazione combinatoria.

Formulazione del Problema

Modello di Programmazione Lineare Intera (PLI)

- **Variabili decisionali**:
  - $x_v = 1$ se il nodo v è incluso nella cricca, 0 altrimenti.
- **Vincoli**:
  - Due nodi non connessi non possono far parte della stessa cricca:

$$x_u + x_v \leq 1 \quad \forall (u, v) \notin E$$

- **Funzione obiettivo**:

$$\max \sum_{v \in V} x_v$$

Euristica del Grado Massimo (Greedy)

Passi:

1. **Inizializzazione**:

- Inizia con una cricca vuota $Q = \{\}$.

2. **Selezione del nodo**:
   - Seleziona il nodo v con il grado più alto e aggiungilo a Q.
3. **Estensione della cricca**:
   - Trova un nodo v non ancora in Q che sia connesso a tutti i nodi in Q e abbia il maggior numero di vicini in comune con i nodi in Q. Aggiungi v a Q.
4. **Terminazione**:
   - Se non è possibile aggiungere ulteriori nodi, restituisci Q.

Esempio: Rete Sociale

Consideriamo la seguente rete sociale:

ID	Nome	Amici con
0	Toi	2, 9, 6
1	Brain	4, 6, 9
2	Annamaria	0, 9, 10
3	Nina	4, 6, 7, 8, 9, 10
4	Walton	1, 3, 5, 6, 7, 8, 9
5	Virgilio	4, 6, 8, 10
6	Teena	0, 1, 3, 4, 5
7	Darrin	3, 4, 9
8	Alessandra	3, 4, 5
9	Harry	0, 1, 2, 3, 4, 7
10	Simona	2, 3, 5

Domande:

1. Trova una cricca massima utilizzando la programmazione intera.
2. Trova tutte le cricche massime utilizzando la programmazione intera.
3. Trova una cricca utilizzando l'euristica del grado massimo.
4. Trova una cricca di peso massimo, dove il peso di un nodo è il suo indice.

Trovare una cricca massima utilizzando la programmazione intera

Modello PLI per la cricca massima

- **Variabili decisionali**:
  $x_i \in \{0, 1\}$ per ogni nodo i, dove $x_i = 1$ se il nodo i è incluso nella cricca.
- **Funzione obiettivo**:

$$\text{Max} \sum_{i=0}^{10} x_i$$

- **Vincoli**:
  Per ogni coppia di nodi (i, j) **non connessi**, aggiungi il vincolo:

$$x_i + x_j \leq 1$$

  Questo garantisce che due nodi non connessi non possano essere entrambi nella cricca.

Applicazione alla rete sociale

- **Nodi non connessi**:
  Ad esempio, Toi (0) non è amico di Brain (1), quindi:

$$x_0 + x_1 \leq 1$$

  Analogamente per tutte le coppie non connesse.

Soluzione
Risolvendo il modello, una **cricca massima** trovata è:
{4 (Walton), 3 (Nina), 9 (Harry)}

- **Connessioni**:
  - Walton (4) è amico di Nina (3) e Harry (9).
  - Nina (3) è amica di Harry (9).

Trovare tutte le cricche massime utilizzando la programmazione intera

Approccio iterativo

1. **Prima cricca**: Risolvi il modello PLI per trovare una cricca massima.
2. **Aggiungi vincoli**: Per escludere la cricca trovata, aggiungi un vincolo del tipo:

$$\sum_{i \in \text{cricca}} x_i \leq |\text{cricca}| - 1$$

3. **Ripeti**: Risolvi nuovamente il modello fino a quando non ci sono più soluzioni.

Cricche massime trovate

1. **{4, 3, 9}**
2. **{4, 3, 8}** (Walton, Nina, Alessandra)
3. **{9, 0, 2}** (Harry, Toi, Annamaria)

Trovare una cricca utilizzando l'euristica del grado massimo

Passi dell'euristica

1. **Calcola i gradi**:

Nodo	Grado
4	7
3	6
9	6
6	5

2. **Seleziona il nodo con grado massimo**: Walton (4).
3. **Filtra i suoi amici**: {1, 3, 5, 6, 7, 8, 9}.
4. **Cerca la cricca più grande tra gli amici comuni**:
   - I nodi {4, 3, 9} sono mutualmente connessi.

Risultato: Cricca **{4, 3, 9}**.

Trovare una cricca di peso massimo (peso = ID del nodo)

Modello PLI modificato

- **Funzione obiettivo**:

$$\text{Max} \sum_{i=0}^{10} i \cdot x_i$$

- **Vincoli**: Stessi vincoli della domanda 1.

Soluzione

La cricca con peso massimo è:

{9 (Harry), 10 (Simona), 5 (Virgilio)}

- **Peso totale**: $9 + 10 + 5 = 24$.
- **Connessioni**:
  - Harry (9) è amico di Simona (10) e Virgilio (5).
  - Simona (10) è amica di Virgilio (5).

Problema della Quasi-Cricca Massima (Maximum Quasi-Clique Problem)

Una **quasi-cricca** è un sottoinsieme di nodi in cui è consentita l'assenza di un numero limitato di archi. Formalmente, una ϵ-quasi-cricca è una cricca a cui mancano al massimo ϵ archi.

Formulazione del Problema

Modello di Programmazione Lineare Intera (PLI)

- **Variabili decisionali**:
  - $x_v = 1$ se il nodo v è incluso nella quasi-cricca, 0 altrimenti.
  - $z_{u,v} = 1$ se i nodi u e v non sono connessi ma fanno parte della quasi-cricca, 0 altrimenti.
- **Vincoli**:
  - Due nodi non connessi non possono far parte della stessa quasi-cricca a meno che non siano un'eccezione:

$$x_u + x_v \leq 1 + z_{u,v} \quad \forall (u,v) \notin E$$

  - Limita il numero di eccezioni:

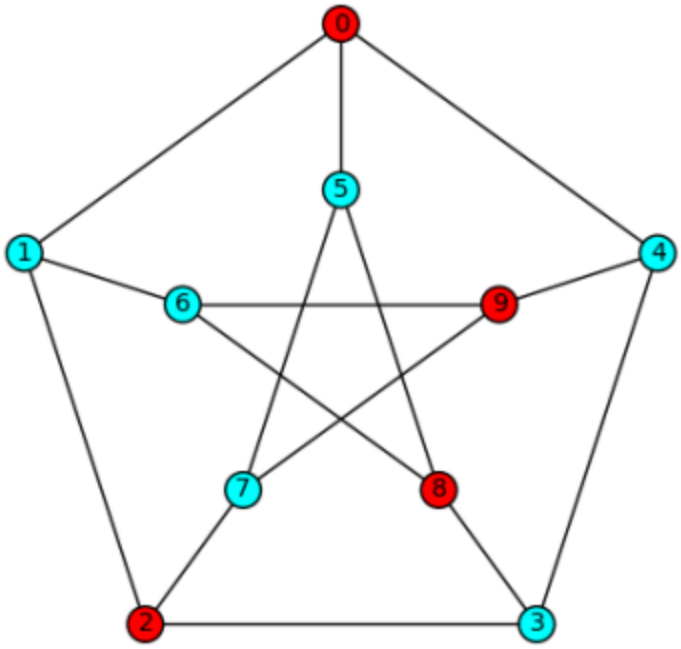
$$\sum_{(u,v)\notin E} z_{u,v} \leq \epsilon$$

- **Funzione obiettivo**:

$$\max \sum_{v \in V} x_v$$

Problema del Massimo Insieme Indipendente (Maximum Independent Set Problem)

Un **insieme indipendente** è un sottoinsieme di nodi in cui nessuna coppia di nodi è connessa da un arco. Il **Problema del Massimo Insieme Indipendente** consiste nel trovare un insieme indipendente di cardinalità massima in un grafo G.
Un insieme indipendente in G è una cricca nel grafo complementare $\overline{G}$. Pertanto, un algoritmo per trovare una cricca massima in $\overline{G}$ risolve anche il problema del massimo insieme indipendente in G.



Formulazione del Problema

Modello di Programmazione Lineare Intera (PLI)

- **Variabili decisionali**:
  - $x_v = 1$ se il nodo v è incluso nell'insieme indipendente, 0 altrimenti.
- **Vincoli**:
  - Due nodi connessi non possono far parte dello stesso insieme indipendente:

$$x_u + x_v \leq 1 \quad \forall (u, v) \in E$$

- **Funzione obiettivo**:

$$\max \sum_{v \in V} x_v$$

Problema della Colorazione dei Grafi (Graph Coloring Problem)

Il **Problema della Colorazione dei Grafi** consiste nell'assegnare colori ai nodi di un grafo non orientato $G = (V, E)$ in modo che due nodi adiacenti non abbiano lo stesso colore, utilizzando il numero minimo di colori possibile. Questo problema è **NP-hard**, il che lo rende estremamente difficile da risolvere in modo ottimale per istanze di grandi dimensioni.

Applicazioni

- **Assegnazione di frequenze**: Assegnazione di frequenze radio senza interferenze.
- **Assegnazione di posti a sedere**: Assegnazione di posti in modo che persone con conflitti non siano vicine.
- **Pianificazione di attività**: Assegnazione di risorse senza sovrapposizioni.
- **Progettazione di esperimenti**: Assegnazione di trattamenti in modo che non ci siano conflitti.
- **Informatica**: Ottimizzazione di algoritmi e strutture dati.

Definizioni Importanti

- **Numero cromatico** $\chi(G)$: Il numero minimo di colori necessari per una colorazione ammissibile di G.
- **Numero di clique** $\omega(G)$: La dimensione della cricca massima in G.
- **Relazione fondamentale**: $\chi(G) \geq \omega(G)$.

Formulazione del Problema

Modello di Programmazione Lineare Intera (PLI)

- **Variabili decisionali**:
  - $x_{v,c} = 1$ se il nodo v è colorato con il colore c, 0 altrimenti.
  - $y_c = 1$ se il colore c è utilizzato, 0 altrimenti.
- **Vincoli**:
  - Due nodi adiacenti non possono avere lo stesso colore:

$$x_{u,c} + x_{v,c} \leq 1 \quad \forall (u, v) \in E,\, c \in C$$

  - Ogni nodo deve essere colorato con un solo colore:

$$\sum_{c \in C} x_{v,c} = 1 \quad \forall v \in V$$

  - Collegamento tra variabili $x_{v,c}$ e y_c:

$$\frac{1}{|V|} \sum_{v \in V} x_{v,c} \leq y_c \quad \forall c \in C$$

- **Funzione obiettivo**:

$$\min \sum_{c \in C} y_c$$

Euristica Greedy

Passi:

1. **Inizializzazione**:
   - Inizia con un grafo non colorato e una lista di colori $C = \{1, 2, \ldots, |V|\}$.
2. **Selezione del nodo**:
   - Scegli un nodo non colorato v.
3. **Assegnazione del colore**:
   - Assegna a v il primo colore disponibile che non è stato usato per i suoi vicini.
4. **Terminazione**:
   - Ripeti i passi 2-3 fino a quando tutti i nodi sono colorati.

Euristica DSatur

Passi:

1. **Inizializzazione**:
   - Inizia con un grafo non colorato e una lista di colori $C = \{1, 2, \ldots, |V|\}$.
2. **Selezione del nodo**:
   - Scegli il nodo non colorato con la **massima saturazione** (numero di colori diversi usati dai vicini) e, in caso di parità, il nodo con il **grado più alto**.
3. **Assegnazione del colore**:
   - Assegna a v il primo colore disponibile che non è stato usato per i suoi vicini.
4. **Terminazione**:
   - Ripeti i passi 2-3 fino a quando tutti i nodi sono colorati.

Esempio: Colorazione di un Grafo

Consideriamo un grafo con 8 nodi e i seguenti archi:

- $E = \{(1,2), (1,3), (2,4), (3,4), (4,5), (5,6), (6,7), (7,8)\}$.

Domande:

1. Trova una colorazione ammissibile utilizzando l'euristica Greedy.

2. Trova una colorazione ammissibile utilizzando l'euristica DSatur.
3. Trova una colorazione ottimale utilizzando la programmazione intera.
4. Aggiungi un numero minimo di archi per aumentare il numero di colori necessari di uno.

Problema di Pianificazione di Progetti con Vincoli di Risorse (RCPSP)

Il **Problema di Pianificazione di Progetti con Vincoli di Risorse (RCPSP)** consiste nel pianificare un insieme di attività (job) in modo da minimizzare la durata totale del progetto (makespan), rispettando i vincoli di precedenza tra le attività e i limiti di disponibilità delle risorse. Questo problema è **NP-hard**, il che lo rende estremamente difficile da risolvere in modo ottimale per istanze di grandi dimensioni.

Definizioni e Input

- **Risorse** $R = \{1, 2, \dots\}$: Risorse rinnovabili con disponibilità per periodo $q_r > 0$.
- **Attività** $J = \{1, 2, \dots\}$: Ogni attività j ha una durata $d_j \geq 0$ e un consumo di risorse $u_{j,r} \geq 0$ per ogni risorsa r.
- **Orizzonte temporale** $T = \{0, 1, 2, \dots\}$: Periodi di tempo disponibili per la schedulazione.
- **Precedenze** $A = \{(i, j)\}$: Coppie di attività dove i deve essere completata prima che j possa iniziare.

Obiettivo

Trovare una schedulazione delle attività che:

1. Rispetti tutte le precedenze.
2. Non superi la disponibilità delle risorse in nessun periodo.
3. Minimizzi la durata totale del progetto (makespan).

Rappresentazione del Problema

- **Diagramma di Gantt**: Rappresentazione grafica della schedulazione delle attività nel tempo.
- **Grafo delle precedenze**: Grafo diretto aciclico (DAG) in cui gli archi rappresentano le relazioni di precedenza.

Formulazione del Problema

Modello di Programmazione Lineare Intera (PLI)

- **Variabili decisionali**:
  - $x_{j,t} = 1$ se l'attività j termina al tempo t, 0 altrimenti.
- **Vincoli**:

- Ogni attività deve essere schedulata esattamente una volta:

$$\sum_{t \in T} x_{j,t} = 1 \quad \forall j \in J$$

- Rispetto delle precedenze:

$$\sum_{t \in T} t \cdot x_{i,t} \leq \sum_{t \in T} t \cdot x_{j,t} - d_j \quad \forall (i,j) \in A$$

- Rispetto della disponibilità delle risorse:

$$\sum_{j \in J} \sum_{t'=t}^{t+d_j-1} u_{j,r} \cdot x_{j,t'} \leq q_r \quad \forall t \in T,\, r \in R$$

- **Funzione obiettivo**:

$$\min \sum_{t \in T} t \cdot x_{n,t}$$

dove n è l'attività finale.

Profilo delle Attività e delle Risorse

- **Profilo delle attività**: Mostra il numero di attività attive in ogni periodo.
- **Profilo delle risorse**: Mostra il consumo di ogni risorsa in ogni periodo.

Algoritmo di Ordinamento Topologico (TopoSort)

Descrizione:
L'algoritmo di ordinamento topologico è un metodo euristico per risolvere il RCPSP. Si basa sull'ordinamento delle attività in base alle precedenze e sulla loro schedulazione nel rispetto dei vincoli di risorse.

Passi:

1. **Inizializzazione**:
   - Crea una schedulazione vuota S.
   - Crea una rete di precedenza ausiliaria $N' = N$.
2. **Ordinamento e schedulazione**:
   - Ripeti fino a quando tutte le attività sono schedulate:
     a) Seleziona un'attività j non ancora schedulata e senza predecessori in N'.

b) Rimuovi j da N'.
c) Schedula j in S al più presto possibile, rispettando i vincoli di risorse e precedenze.

Esempio:
Consideriamo il seguente progetto:

- Attività: A (2 giorni), B (3 giorni), C (1 giorno), D (4 giorni), E (5 giorni), F (1 giorno).
- Precedenze: A → D, B → F, C → E, D → E.
- Risorse: 4 persone, 5 costi.

Schedulazione con TopoSort:

1. Schedula A al tempo 0.
2. Schedula B al tempo 0.
3. Schedula C al tempo 0.
4. Schedula D al tempo 2 (dopo A).
5. Schedula F al tempo 3 (dopo B).
6. Schedula E al tempo 6 (dopo C e D).

Euristica del Punto Alfa (α-Point Heuristic)

L'euristica del punto alfa utilizza il rilassamento lineare (LP) del problema RCPSP per determinare un ordinamento delle attività basato su valori frazionari $x_{j,t}$. L'obiettivo è trovare una schedulazione approssimata che rispetti i vincoli di precedenza e risorse.

Passi:

1. **Risoluzione del rilassamento lineare (LP)**:
   - Risolvi il rilassamento lineare del modello RCPSP per ottenere valori frazionari $x_{j,t}$.
2. **Calcolo dei punti alfa**:
   - Per ogni attività j, calcola il punto alfa t tale che $\sum_{t'=0}^{t} x_{j,t'} \geq \alpha$.
3. **Ordinamento e schedulazione**:
   - Ordina le attività in base ai punti alfa e schedula rispettando i vincoli di precedenza e risorse.

Esempio:
Utilizzando lo stesso progetto dell'esempio precedente:

- Risolvi il LP per ottenere i valori $x_{j,t}$.
- Calcola i punti alfa per $\alpha = 0.5$.
- Ordina le attività in base ai punti alfa e schedula rispettando i vincoli.

Esempi di Applicazione

Esempio 1: Progetto di Costruzione

- **Attività**: A (2 giorni), B (3 giorni), C (1 giorno), D (4 giorni), E (5 giorni), F (1 giorno).
- **Precedenze**: A → D, B → F, C → E, D → E.
- **Risorse**: 4 persone, 5 costi.

Schedulazione con TopoSort:

1. A inizia al tempo 0.
2. B inizia al tempo 0.
3. C inizia al tempo 0.
4. D inizia al tempo 2.
5. F inizia al tempo 3.
6. E inizia al tempo 6.

Schedulazione con α-Point Heuristic:

- Risolvi il LP e calcola i punti alfa.
- Ordina le attività e schedula rispettando i vincoli.

Esempio: Costruzione di un Ufficio

Consideriamo il seguente progetto di ristrutturazione di un ufficio:

Attività	Simbolo	Precedenza	Durata	Persone	Costi (in 1000)
Preparare opzioni di finanziamento	A	-	2	3	3
Preparare schizzi preliminari	B	-	3	2	1
Delineare le specifiche	C	-	1	1	3
Preparare disegni	D	A	4	3	4
Scrivere le specifiche	E	C, D	5	3	1
Eseguire le stampe	F	B	1	1	1

Disponibilità delle risorse:

- $q_{Persone} = 4$
- $q_{Costi} = 5$

Esercizi

1. **Costruzione del Modello in Excel**:
   - Costruire un modello in Excel che minimizzi il makespan utilizzando Excel Solver.
   - Visualizzare il profilo delle attività.
   - Visualizzare il profilo delle risorse.
2. **Risoluzione con MiniZinc**:
   - Trovare una schedulazione ottimale utilizzando la programmazione a vincoli (CP) in MiniZinc.
   - Visualizzare la schedulazione utilizzando un diagramma di Gantt.
   - Visualizzare i profili delle attività e delle risorse.
3. **Riduzione della Disponibilità delle Risorse**:
   - Determinare di quanto aumenta il makespan se la disponibilità della risorsa 1 (Persone) viene ridotta a 5.
   - Verificare i risultati visivamente.

Estensioni e Varianti del RCPSP

Attività con Modalità Multiple (Multi-Mode RCPSP)

- **Modalità**: Ogni attività può essere eseguita in una delle sue M_j modalità, ciascuna con una durata p_{jm} e un consumo di risorse r_{jmk}.
- **Risorse non rinnovabili**: Oltre alle risorse rinnovabili, possono essere considerate risorse non rinnovabili (ad esempio, un budget totale).

Vincoli Temporali Generalizzati

- **Lagh temporali minimi e massimi**: Possono essere specificati tempi minimi o massimi tra il completamento di un'attività e l'inizio di un'altra.
- **Date di rilascio e scadenza**: Le attività possono avere date di inizio e fine specifiche.

Risorse Parzialmente Rinnovabili

- **Disponibilità variabile**: La disponibilità delle risorse può variare nel tempo o essere limitata a sottoinsiemi di periodi.

Obiettivi Alternativi

- **Minimizzazione dei costi**: Oltre al makespan, possono essere considerati obiettivi come la minimizzazione dei costi totali o il massimizzazione del valore attuale netto (NPV).
- **Robustezza**: Creare schedule robusti che minimizzino l'impatto di ritardi imprevisti.

Algoritmi e Metodi di Soluzione

Programmazione Lineare Intera (PLI)

- **Formulazione**: Il problema è formulato come un modello di programmazione lineare intera, con variabili binarie che indicano l'inizio delle attività in specifici periodi.
- **Vincoli**: Includono precedenze, disponibilità delle risorse e obiettivi.

Programmazione a Vincoli (CP)

- **Approccio**: Utilizza variabili di intervallo e vincoli cumulativi per modellare le attività e le risorse.
- **Vantaggi**: Efficace per problemi con vincoli complessi e istanze di grandi dimensioni.

Algoritmi Ibridi

- **Combinazione di PL e CP**: Utilizza il rilassamento lineare per ottenere limiti superiori e euristiche basate su list scheduling per generare soluzioni fattibili.
- **Riduzione del problema**: Tecniche di riduzione per diminuire le dimensioni delle istanze.

Applicazioni

Il **Problema di Pianificazione di Progetti con Vincoli di Risorse (RCPSP)** è un problema di ottimizzazione complesso che consiste nel pianificare un insieme di attività (job) rispettando i vincoli di precedenza e la disponibilità delle risorse, con l'obiettivo di minimizzare la durata totale del progetto (makespan). Questo problema è **NP-hard**, il che lo rende estremamente difficile da risolvere in modo ottimale per istanze di grandi dimensioni.

Outline della Presentazione

1. **Traversate di Navi nei Porti Marittimi**:
   - Modellazione del problema come RCPSP.
   - Approccio di programmazione intera e risultati computazionali.
2. **Pianificazione Strategica di Miniere Sotterranee**:
   - Modellazione del problema come RCPSP con flussi di cassa scontati (RCPSP+DC).
   - Algoritmi ibridi di programmazione matematica e programmazione a vincoli.
3. **Risultati Computazionali**:
   - Confronto tra approcci e prestazioni.
4. **Conclusioni e Prospettive Future**.

Traversate di Navi nei Porti Marittimi

Problema di Pianificazione delle Traversate (WSSP)

- **Input**:

    - **Porti e vie d'acqua**: Larghezza, profondità, traffico bidirezionale.
    - **Navi in arrivo e partenza**: Tempi stimati di arrivo (ETA) e partenza (ETD).
- **Obiettivo**:
    - Minimizzare il tempo totale di turnaround delle navi.
    - Ridurre i tempi di attesa, la congestione del traffico e le emissioni.
- **Vincoli**:
    - Geografici: Larghezza e profondità delle vie d'acqua.
    - Operativi: Traffico opposto, distanza di sicurezza.

Modellazione come RCPSP Multi-Modal (MM-RCPSP)

- **Job**: Navi in arrivo e partenza.
- **Modalità**: Vie d'acqua disponibili.
- **Risorse**: Larghezza e profondità delle vie d'acqua, traffico parallelo.
- **Precedenze**: Vincoli temporali tra le traversate.

Risultati Computazionali

- **Istanza**: 40 casi basati sul porto di Shanghai.
- **Tempo di discretizzazione**: 30 minuti.
- **Risultati**:
    - Tutte le istanze risolte in modo ottimale.
    - 82% risolte al nodo radice.
    - Meno di 5 nodi di branching.

Pianificazione Strategica di Miniere Sotterranee

Problema di Pianificazione delle Miniere

- **Obiettivo**:
    - Trovare un piano operativo a lungo termine che massimizzi il profitto netto scontato (NPV).
    - Attività: Scavo, trasporto, riempimento, ecc.
    - Vincoli: Capacità del mulino, stabilità, sicurezza, ventilazione.
- **Sfide**:
    - Modelli molto grandi (fino a 30.000 attività e 50 anni di pianificazione).
    - Vincoli di produzione complessi.

Modellazione come RCPSP con Flussi di Cassa Scontati (RCPSP+DC)

- **Job**: Attività minerarie.
- **Risorse**: Capacità del mulino, disponibilità di macchinari/personale.

- **Precedenze**: Sviluppo della struttura della miniera nel tempo.
- **Obiettivo**:
  - Massimizzare il valore attuale netto (NPV) dei flussi di cassa.

Algoritmi Ibridi

1. **Riduzione del Problema**:
   - Tecniche di riduzione per diminuire le dimensioni delle istanze.
2. **Risoluzione del Rilassamento Lineare**:
   - Algoritmo di Bienstock-Zuckerberg per ottenere un limite superiore.
3. **Euristiche di List Scheduling**:
   - Assegnazione di priorità basata sulle soluzioni LP.
4. **Programmazione a Vincoli (CP)**:
   - Utilizzo di IBM ILOG CP Optimizer per ottimizzare ulteriormente.

Risultati Computazionali

- **Istanza**: 16 casi accademici e reali.
- **Risultati**:
  - Riduzione significativa dei tempi di esecuzione.
  - Miglioramento dei gap di ottimalità.
  - Soluzioni fattibili per istanze di grandi dimensioni.

Confronto tra Approcci

- **Programmazione Lineare (LP)**:
  - Risoluzione efficiente del rilassamento lineare.
- **Programmazione a Vincoli (CP)**:
  - Miglioramento delle soluzioni iniziali.
- **Algoritmi Ibridi**:
  - Combinazione di LP e CP per ottenere soluzioni ottimali o quasi ottimali.

Prestazioni

- **Tempi di Esecuzione**:
  - Riduzione significativa rispetto agli approcci tradizionali.
- **Gap di Ottimalità**:
  - Miglioramento fino al 20-40% rispetto alle euristiche esistenti.

Conclusioni

- **Traversate di Navi**:

    - La modellazione come MM-RCPSP consente di risolvere problemi realistici in modo efficiente.
- **Pianificazione di Miniere**:
    - Gli algoritmi ibridi LP-CP sono efficaci per istanze di grandi dimensioni.

Prospettive Future

- **Integrazione di Operazioni di Attracco**:
    - Estendere il modello per includere le operazioni di attracco.
- **Sviluppo di Tagli e Euristiche**:
    - Migliorare ulteriormente le prestazioni degli algoritmi.
- **Applicazione della Programmazione a Vincoli**:
    - Esplorare ulteriori applicazioni della CP in contesti minerari.