Lezione di Informatica Teorica: NP-Completezza
Exact Cover e Knapsack

Appunti da Trascrizione Automatica

24 Aprile 2024

Indice

1 Introduzione alle Classi di Complessità

Questa lezione conclude l'introduzione esplicita alla classe NP, prima di esplorare altre classi di complessità come la complessità spaziale e le classi di funzioni (problemi di calcolo piuttosto che di decisione). Verrà inoltre dimostrato il Teorema di Cook nella prossima lezione, che stabilisce che SAT è NP-completo.

1.1 Definizioni Fondamentali

Definizione 1 (Problema NP-Completo)**.** *Un problema di decisione L è NP-completo se:*

1. *$L \in NP$ (appartiene alla classe NP).*
2. *L è NP-hard (almeno "duro" quanto tutti i problemi in NP), ovvero ogni problema $L' \in NP$ è riducibile a L in tempo polinomiale ($L' \leq_P L$).*

Definizione 2 (Problema NP-Hard)**.** *Un problema L è NP-hard se ogni problema $L' \in NP$ è riducibile a L in tempo polinomiale ($L' \leq_P L$).*

È importante sottolineare che le classi P e NP, e i concetti di NP-hard e NP-completo, si applicano esclusivamente ai **problemi di decisione**.

Esempio 1. ***Problema di Somma:***

- ***Problema di calcolo:*** *"Sommare due numeri". Questo non è un problema di decisione e quindi non rientra nelle classi P o NP.*
- ***Problema di decisione:*** *"Dati tre numeri a, b, c, è vero che $c = a + b$?" Questo è un problema di decisione e, poiché risolvibile in tempo polinomiale, rientra nella classe P.*

Oggi analizzeremo due problemi NP-completi: *Exact Cover* e *Knapsack*.

2 Exact Cover

Definizione 3 (Exact Cover)**.** ***Input:*** *Una coppia (U, F) dove:*

- *$U = \{u_1, u_2, \dots, u_N\}$ è un insieme finito di oggetti, chiamato* universo*.*
- *$F = \{S_1, S_2, \dots, S_M\}$ è una famiglia di sottoinsiemi di U, ovvero $S_j \subseteq U$ per ogni $j \in \{1, \dots, M\}$.*

Domanda: *Esiste un sottoinsieme $F' \subseteq F$ tale che gli insiemi in F' formano una* partizione *di U? Una partizione di U significa che gli insiemi in F' sono a due a due disgiunti (cioè $S_a \cap S_b = \emptyset$ per ogni $S_a, S_b \in F', a \neq b$) e la loro unione è uguale a U (cioè $\bigcup_{S \in F'} S = U$).*

2.1 Membership in NP

Exact Cover è in NP. Per dimostrarlo, è sufficiente mostrare che data un'istanza YES, possiamo verificare la soluzione in tempo polinomiale.

- **Guess (Indovina):** Un certificato per un'istanza YES di Exact Cover è il sottoinsieme $F' \subseteq F$ che si afferma essere la partizione.

- **Check (Verifica):** Data F', possiamo verificare in tempo polinomiale se:
  1. Tutti gli insiemi in F' sono a due a due disgiunti.
  2. L'unione di tutti gli insiemi in F' è uguale all'universo U.

  Queste verifiche possono essere eseguite in tempo polinomiale rispetto alla dimensione dell'input.

2.2 Hardness (Riduzione da 3-SAT)

Dimostriamo che Exact Cover è NP-hard riducendolo da 3-SAT (che sappiamo essere NP-completo). Sia ϕ un'istanza di 3-SAT. ϕ è una formula booleana in forma normale congiuntiva (CNF), composta da L clausole $C_1, \ldots, C_L$, dove ogni clausola C_j contiene esattamente 3 letterali: $C_j = (\lambda_{j,1} \vee \lambda_{j,2} \vee \lambda_{j,3})$. Sia N il numero di variabili in ϕ. Dobbiamo costruire una funzione polinomiale f che trasforma ϕ in un'istanza (U_ϕ, F_ϕ) di Exact Cover tale che ϕ è soddisfacibile se e solo se (U_ϕ, F_ϕ) è un'istanza YES di Exact Cover.

2.2.1 Costruzione dell'Istanza (U_ϕ, F_ϕ)

1. Universo U_ϕ: L'universo U_ϕ è costruito per rappresentare le variabili, le clausole e i letterali della formula ϕ. $U_\phi = \{\text{var}_1, \ldots, \text{var}_N\} \cup \{c_1, \ldots, c_L\} \cup \{l_{j,k} \mid j \in [1, L], k \in [1, 3]\}$

- var_i: Un oggetto per ogni variabile x_i in ϕ.
- c_j: Un oggetto per ogni clausola C_j in ϕ.
- $l_{j,k}$: Un oggetto per ogni letterale $\lambda_{j,k}$ in ϕ.

2. Famiglia di Sottoinsiemi F_ϕ: La famiglia F_ϕ contiene diversi tipi di insiemi, progettati per simulare l'assegnazione di verità e la soddisfazione delle clausole.

1. **Insiemi di assegnamento variabile (Type 1):** Per ogni variabile x_i ($i \in [1, N]$), creiamo due insiemi:
   - $T_i^{true} = \{\text{var}_i\} \cup \{l_{j,k} \mid \text{il letterale } \lambda_{j,k} \text{ è } \neg x_i\}$ (Questo insieme "copre" l'oggetto var_i e tutti gli oggetti $l_{j,k}$ corrispondenti a letterali che diventerebbero falsi se x_i fosse assegnata a TRUE).
   - $T_i^{false} = \{\text{var}_i\} \cup \{l_{j,k} \mid \text{il letterale } \lambda_{j,k} \text{ è } x_i\}$ (Questo insieme "copre" l'oggetto var_i e tutti gli oggetti $l_{j,k}$ corrispondenti a letterali che diventerebbero falsi se x_i fosse assegnata a FALSE).
2. **Insiemi di soddisfazione clausola (Type 2):** Per ogni clausola $C_j = (\lambda_{j,1} \vee \lambda_{j,2} \vee \lambda_{j,3})$ ($j \in [1, L]$), creiamo tre insiemi:
   - $S_{j,1} = \{c_j, l_{j,1}\}$
   - $S_{j,2} = \{c_j, l_{j,2}\}$
   - $S_{j,3} = \{c_j, l_{j,3}\}$

   (Questi insiemi rappresentano la soddisfazione della clausola C_j tramite uno dei suoi letterali. Un solo insieme di questo tipo può essere scelto per ogni c_j nella partizione).

3. **Insiemi di pulizia letterale (Type 3):** Per ogni oggetto letterale $l_{j,k}$ in U_ϕ, creiamo un insieme singleton:

    - $\{l_{j,k}\}$

    (Questi insiemi sono usati per "raccogliere le briciole", cioè per coprire gli oggetti $l_{j,k}$ che non sono stati coperti dagli insiemi di Tipo 1 o Tipo 2 selezionati per la partizione. Sono usati come elementi di "riserva").

Esempio 2 (Costruzione di (U_ϕ, F_ϕ))**.** *Sia* $\phi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$. *Qui* $N = 4$ *(variabili* $x_1, \ldots, x_4$*) e* $L = 2$ *(clausole* C_1, C_2*).*

Universo U_ϕ***:***

- *Variabili:* $\{var_1, var_2, var_3, var_4\}$
- *Clausole:* $\{c_1, c_2\}$
- *Letterali:* $\{l_{1,1}, l_{1,2}, l_{1,3}, l_{2,1}, l_{2,2}, l_{2,3}\}$ *(corrispondenti a* $x_1, \neg x_2, x_3$ *per* C_1 *e* $\neg x_1, x_2, x_4$ *per* C_2*).*

Quindi, $U_\phi = \{var_1, var_2, var_3, var_4, c_1, c_2, l_{1,1}, l_{1,2}, l_{1,3}, l_{2,1}, l_{2,2}, l_{2,3}\}$.

Famiglia F_ϕ***:***

- ***Tipo 1 (assegnamento variabile):***
    - $T_1^{true} = \{var_1\} \cup \{l_{2,1} \text{ (per } \neg x_1 \text{ in } C_2)\}$
    - $T_1^{false} = \{var_1\} \cup \{l_{1,1} \text{ (per } x_1 \text{ in } C_1)\}$
    - $T_2^{true} = \{var_2\} \cup \{l_{1,2} \text{ (per } \neg x_2 \text{ in } C_1)\}$
    - $T_2^{false} = \{var_2\} \cup \{l_{2,2} \text{ (per } x_2 \text{ in } C_2)\}$
    - $T_3^{true} = \{var_3\} \cup \emptyset$
    - $T_3^{false} = \{var_3\} \cup \{l_{1,3} \text{ (per } x_3 \text{ in } C_1)\}$
    - $T_4^{true} = \{var_4\} \cup \emptyset$
    - $T_4^{false} = \{var_4\} \cup \{l_{2,3} \text{ (per } x_4 \text{ in } C_2)\}$
- ***Tipo 2 (soddisfazione clausola):***
    - $S_{1,1} = \{c_1, l_{1,1}\}$
    - $S_{1,2} = \{c_1, l_{1,2}\}$
    - $S_{1,3} = \{c_1, l_{1,3}\}$
    - $S_{2,1} = \{c_2, l_{2,1}\}$
    - $S_{2,2} = \{c_2, l_{2,2}\}$
    - $S_{2,3} = \{c_2, l_{2,3}\}$
- ***Tipo 3 (pulizia letterale):***
    - $\{l_{1,1}\}, \{l_{1,2}\}, \{l_{1,3}\}, \{l_{2,1}\}, \{l_{2,2}\}, \{l_{2,3}\}$

2.2.2 Dimostrazione dell'Equivalenza

Teorema 1. *La formula ϕ è soddisfacibile se e solo se l'istanza di Exact Cover (U_ϕ, F_ϕ) ha una partizione.*

Dimostrazione. **Parte 1: Se ϕ è soddisfacibile $\implies$ (U_ϕ, F_ϕ) ha una partizione.** Supponiamo che ϕ sia soddisfacibile. Allora esiste un assegnamento di verità $\sigma : \{x_1, \dots, x_N\} \to \{\text{TRUE}, \text{FALSE}\}$ tale che ϕ è TRUE. Costruiamo una partizione $P \subseteq F_\phi$ come segue:

1. Per ogni variabile x_i:
   - Se $\sigma(x_i) = \text{TRUE}$, includiamo T_i^{true} in P.
   - Se $\sigma(x_i) = \text{FALSE}$, includiamo T_i^{false} in P.

   Questi insiemi coprono tutti gli oggetti var_i esattamente una volta. Inoltre, coprono alcuni oggetti $l_{j,k}$ (quelli che vengono falsificati dall'assegnamento σ).

2. Per ogni clausola C_j: Poiché ϕ è soddisfacibile, C_j è soddisfatta da σ. Ciò significa che almeno uno dei letterali in C_j è TRUE sotto σ. Scegliamo esattamente uno di questi letterali $\lambda_{j,k'}$ che rende C_j vera, e includiamo l'insieme $S_{j,k'} = \{c_j, l_{j,k'}\}$ in P. Questi insiemi coprono tutti gli oggetti c_j esattamente una volta. Essi coprono anche un oggetto $l_{j,k'}$ per ogni clausola. Questi $l_{j,k'}$ sono oggetti che corrispondono a letterali veri sotto σ, e quindi non sono stati coperti dagli insiemi di Tipo 1.

3. Per tutti gli oggetti $l_{j,k}$ rimanenti (quelli non coperti dagli insiemi di Tipo 1 o Tipo 2), includiamo il loro singleton $\{l_{j,k}\}$ in P.

Verifichiamo che P è una partizione:

- **Disgiunzione:**
  - Gli insiemi di Tipo 1 (assegnamento variabile) sono disgiunti tra loro per lo stesso var_i perché solo uno T_i^{true} o T_i^{false} è scelto. Per $\text{var}_i \neq \text{var}_k$, essi non condividono var oggetti. Possono condividere oggetti $l_{j,k}$, ma non è un problema per la disgiunzione complessiva perché l'insieme selezionato sarà un sottoinsieme della partizione finale.
  - Gli insiemi di Tipo 2 (soddisfazione clausola) sono disgiunti tra loro per la stessa c_j perché solo uno $S_{j,k'}$ è scelto. Per $c_j \neq c_k$, non condividono c oggetti. Possono condividere l oggetti, ma questo è gestito dal processo di selezione e dalla gestione dei singleton.
  - Gli insiemi di Tipo 1 e Tipo 2 non condividono oggetti var_i o c_j. Possono condividere oggetti $l_{j,k}$. Per costruzione, gli $l_{j,k}$ in T_i^{true} o T_i^{false} sono quelli corrispondenti a letterali falsi sotto σ. Gli $l_{j,k'}$ in $S_{j,k'}$ sono quelli corrispondenti a letterali veri sotto σ. Dunque, un oggetto $l_{j,k}$ non può essere presente sia in un T_i selezionato che in un S_j selezionato.
  - I singleton $\{l_{j,k}\}$ sono usati solo per coprire gli $l_{j,k}$ che non sono stati inclusi in nessun T_i o S_j selezionato.

- **Unione:**
  - Gli oggetti var_i sono coperti esattamente una volta da T_i^{true} o T_i^{false}.
  - Gli oggetti c_j sono coperti esattamente una volta da $S_{j,k'}$ (uno per clausola).

– Gli oggetti $l_{j,k}$ sono coperti: se falsificati da σ, sono inclusi in un T_i selezionato; se rendono vera una clausola C_j e vengono scelti per la partizione, sono inclusi in un $S_{j,k'}$ selezionato; altrimenti, sono coperti dal loro singleton $\{l_{j,k}\}$. Poiché ogni $l_{j,k}$ è o falsificato o vero (e se vero, può essere scelto per una clausola), tutti gli $l_{j,k}$ sono coperti.

Pertanto, P è una partizione di U_ϕ.

Parte 2: Se (U_ϕ, F_ϕ) ha una partizione $\implies$ ϕ è soddisfacibile. Supponiamo che esista una partizione $P \subseteq F_\phi$ di U_ϕ. Definiamo un assegnamento di verità σ per le variabili di ϕ: Per ogni variabile x_i:

– Se $T_i^{true} \in P$, allora $\sigma(x_i) = \text{TRUE}$.

– Se $T_i^{false} \in P$, allora $\sigma(x_i) = \text{FALSE}$.

Verifica di σ:

– **Ogni variabile ha un assegnamento:** Ogni oggetto $\text{var}_i \in U_\phi$ deve essere coperto da un insieme in P. L'unico modo per coprire var_i è includere o T_i^{true} o T_i^{false} in P. Poiché P è una partizione, non possono essere entrambi in P (condividerebbero var_i), quindi esattamente uno è in P. Ciò assicura che σ assegna un valore a ogni variabile.

– **Assegnamento consistente:** Poiché solo uno tra T_i^{true} e T_i^{false} può essere in P, σ non assegna contemporaneamente TRUE e FALSE a nessuna variabile.

Soddisfazione di ϕ: Per dimostrare che σ soddisfa ϕ, dobbiamo mostrare che ogni clausola C_j è TRUE sotto σ. Per ogni oggetto $c_j \in U_\phi$, esso deve essere coperto da un insieme in P. Gli unici insiemi in F_ϕ che contengono c_j sono $S_{j,1}, S_{j,2}, S_{j,3}$. Dunque, esattamente uno di questi insiemi, diciamo $S_{j,k'} = \{c_j, l_{j,k'}\}$, deve essere in P. Questo significa che il letterale $\lambda_{j,k'}$ corrispondente a $l_{j,k'}$ deve essere TRUE sotto l'assegnamento σ. Se $\lambda_{j,k'}$ fosse FALSE sotto σ, allora l'oggetto $l_{j,k'}$ sarebbe incluso in uno degli insiemi T_i^{true} o T_i^{false} selezionati (poiché $l_{j,k'}$ è falsificato). Ma se $l_{j,k'}$ fosse in un T_i selezionato e anche in $S_{j,k'}$ (che è in P), ci sarebbe un'intersezione tra T_i e $S_{j,k'}$, violando la proprietà di disgiunzione di una partizione. Pertanto, $\lambda_{j,k'}$ deve essere TRUE sotto σ, il che significa che la clausola C_j è soddisfatta. Poiché questo vale per ogni C_j, ϕ è soddisfacibile.

$\square$

Conclusione: Exact Cover è NP-completo.

3 Knapsack (Problema della Bisaccia)

3.1 Definizione dell'ottimizzazione e della decisione

Definizione 4 (Knapsack (Versione Ottimizzazione))**.** ***Input:*** *Un insieme di N oggetti, per ognuno dei quali:*

- w_i*: un peso associato.*
- v_i*: un valore associato.*

Una capacità massima W (peso della bisaccia). ***Domanda:*** *Trovare un sottoinsieme di oggetti $S \subseteq \{1, \dots, N\}$ tale che la somma dei pesi degli oggetti in S non superi W ($\sum_{i \in S} w_i \leq W$) e la somma dei valori degli oggetti in S sia massimizzata ($\sum_{i \in S} v_i$ sia massima).*

La versione di ottimizzazione di Knapsack non è direttamente in NP, poiché NP è una classe di problemi di decisione. Per studiarlo nella teoria della complessità, usiamo la versione di decisione:

Definizione 5 (Knapsack (Versione Decisione))**.** ***Input:*** *Un insieme di N oggetti, per ognuno dei quali:*

- *w_i: un peso associato.*
- *v_i: un valore associato.*

Una capacità massima W (peso della bisaccia) e un valore soglia K. ***Domanda:*** *Esiste un sottoinsieme di oggetti $S \subseteq \{1, \dots, N\}$ tale che la somma dei pesi degli oggetti in S non superi W ($\sum_{i \in S} w_i \leq W$) e la somma dei valori degli oggetti in S sia almeno K ($\sum_{i \in S} v_i \geq K$)?*

3.2 Membership in NP (Versione Decisione)

Knapsack (versione decisione) è in NP.

- **Guess (Indovina):** Un certificato per un'istanza YES di Knapsack è il sottoinsieme $S \subseteq \{1, \dots, N\}$ degli oggetti scelti.
- **Check (Verifica):** Dati S, possiamo verificare in tempo polinomiale se:
  1. La somma dei pesi $\sum_{i \in S} w_i \leq W$.
  2. La somma dei valori $\sum_{i \in S} v_i \geq K$.

  Queste verifiche sono semplici sommatorie e confronti, eseguibili in tempo polinomiale.

3.3 Hardness (Riduzione da Exact Cover)

Dimostriamo che Knapsack (decisione) è NP-hard riducendolo da Exact Cover. Sia (U, F) un'istanza di Exact Cover, con $U = \{u_1, \dots, u_N\}$ e $F = \{S_1, \dots, S_M\}$. Dobbiamo costruire una funzione polinomiale f che trasforma (U, F) in un'istanza di Knapsack (oggetti, pesi w_i, valori v_i, capacità W, soglia K) tale che (U, F) è un'istanza YES di Exact Cover se e solo se l'istanza di Knapsack generata è un'istanza YES.

3.3.1 Costruzione dell'Istanza di Knapsack

La riduzione sfrutta una particolare forma dell'istanza di Knapsack.

1. Assunzione Specifiche: Costruiremo un'istanza di Knapsack in cui:

- I valori sono uguali ai pesi: $v_i = w_i$ per ogni oggetto i.
- La soglia K è uguale alla capacità W: $K = W$.

Con queste assunzioni, il problema di decisione del Knapsack diventa: "Esiste un sottoinsieme di oggetti S tale che $\sum_{i \in S} w_i = W$?" (poiché $\sum w_i \leq W$ e $\sum v_i \geq K$ con $v_i = w_i$ e $K = W$ implica $\sum w_i \geq W$, quindi uguaglianza).

2. Oggetti della Bisaccia: Ci sono M oggetti nella bisaccia, uno per ogni insieme $S_j \in F$.

3. Pesi (w_j) e Valori (v_j): Per ogni oggetto j (corrispondente al set $S_j \in F$), definiamo il suo peso w_j (e valore $v_j = w_j$) in modo che codifichi la composizione del set S_j. Per evitare i problemi di "riporto" che si avrebbero con la rappresentazione binaria standard, useremo una base numerica sufficientemente grande, in particolare $M + 1$ (dove M è il numero totale di insiemi in F). Questo assicura che la somma di M cifre 0 o 1 in qualsiasi posizione non genererà un riporto.

Il peso w_j (e valore v_j) per l'oggetto j (che rappresenta l'insieme $S_j \in F$) è definito come:

$$w_j = \sum_{k=1}^{N} \delta_{j,k} \cdot (M+1)^{N-k}$$

dove:

- N è il numero di elementi nell'universo U.
- M è il numero di insiemi nella famiglia F.
- $\delta_{j,k} = 1$ se l'elemento $u_k \in U$ è contenuto nell'insieme S_j (cioè $u_k \in S_j$).
- $\delta_{j,k} = 0$ se l'elemento $u_k \in U$ non è contenuto nell'insieme S_j (cioè $u_k \notin S_j$).

Questa formula interpreta w_j come un numero in base $(M + 1)$, dove la k-esima cifra (da sinistra, corrispondente a u_k) è 1 se $u_k \in S_j$ e 0 altrimenti.

4. Capacità W e Soglia K: La capacità della bisaccia W (e la soglia K) è definita come il numero la cui rappresentazione in base $(M + 1)$ è composta da tutti '1'. Questo significa che ogni elemento dell'universo deve essere coperto esattamente una volta.

$$W = K = \sum_{k=1}^{N} 1 \cdot (M+1)^{N-k}$$

Esempio 3 (Knapsack Reduction (Base $M + 1$))**.** *Sia* $U = \{u_1, u_2, u_3, u_4\}$ *(*$N = 4$*) e* $F = \{S_1, S_2, S_3\}$ *(*$M = 3$*).*

- $S_1 = \{u_3, u_4\}$
- $S_2 = \{u_2, u_4\}$
- $S_3 = \{u_2, u_3, u_4\}$

La base sarà $M + 1 = 3 + 1 = 4$. *I pesi* w_j *(e valori* v_j*) sono:*

- $w_1 = (0011)_4 = 0 \cdot 4^3 + 0 \cdot 4^2 + 1 \cdot 4^1 + 1 \cdot 4^0 = 4 + 1 = 5$
- $w_2 = (0101)_4 = 0 \cdot 4^3 + 1 \cdot 4^2 + 0 \cdot 4^1 + 1 \cdot 4^0 = 16 + 1 = 17$
- $w_3 = (0111)_4 = 0 \cdot 4^3 + 1 \cdot 4^2 + 1 \cdot 4^1 + 1 \cdot 4^0 = 16 + 4 + 1 = 21$

La capacità e soglia $W = K$ *sono:*

- $W = K = (1111)_4 = 1 \cdot 4^3 + 1 \cdot 4^2 + 1 \cdot 4^1 + 1 \cdot 4^0 = 64 + 16 + 4 + 1 = 85$

In questo esempio, l'istanza di Exact Cover è NO (l'elemento u_1 *non è coperto da nessun set in F). Se la riduzione funziona correttamente, l'istanza di Knapsack generata dovrebbe essere anch'essa NO (non dovrebbe essere possibile sommare* w_j *a 85).*

3.3.2 Dimostrazione dell'Equivalenza

Teorema 2. *L'istanza di Exact Cover (U, F) ha una partizione se e solo se l'istanza di Knapsack costruita ha una soluzione con peso totale W e valore totale K.*

Dimostrazione. Ricordiamo che, per costruzione, il problema Knapsack si riduce a trovare un sottoinsieme di pesi che sommi esattamente a W, poiché $v_j = w_j$ e $K = W$.

Parte 1: Se (U, F) ha una partizione $\implies$ l'istanza di Knapsack ha una soluzione. Supponiamo che esista un sottoinsieme $F' \subseteq F$ tale che F' è una partizione di U. Questo significa che:

1. Gli insiemi in F' sono a due a due disgiunti: $S_a \cap S_b = \emptyset$ per ogni $S_a, S_b \in F', a \neq b$.
2. La loro unione è U: $\bigcup_{S \in F'} S = U$.

Consideriamo il sottoinsieme di oggetti Knapsack corrispondente agli insiemi in F'. Siano questi oggetti $S'_{items} = \{j \mid S_j \in F'\}$. Calcoliamo la somma dei pesi di questi oggetti: $\sum_{j \in S'_{items}} w_j$. Per la definizione di w_j e le proprietà di F':

$$\sum_{j \in S'_{items}} w_j = \sum_{j \in S'_{items}} \left(\sum_{k=1}^{N} \delta_{j,k} \cdot (M+1)^{N-k} \right)$$

Invertendo l'ordine delle sommatorie:

$$\sum_{k=1}^{N} \left(\sum_{j \in S'_{items}} \delta_{j,k} \right) \cdot (M+1)^{N-k}$$

Poiché F' è una partizione di U:

- Ogni elemento $u_k \in U$ appartiene a *esattamente uno* degli insiemi in F'. Questo significa che per ogni $k \in \{1, \ldots, N\}$, la somma $\sum_{j \in S'_{items}} \delta_{j,k}$ sarà esattamente 1. (Non può essere più di 1 perché i set sono disgiunti; non può essere meno di 1 perché la loro unione è U).

Quindi, la somma dei pesi diventa:

$$\sum_{k=1}^{N} 1 \cdot (M+1)^{N-k}$$

Questa è esattamente la definizione di W.

$$\sum_{j \in S'_{items}} w_j = W$$

Poiché per costruzione $v_j = w_j$ e $K = W$, anche $\sum_{j \in S'_{items}} v_j = K$. Dunque, se (U, F) ha una partizione, l'istanza di Knapsack ha una soluzione.

Parte 2: Se l'istanza di Knapsack ha una soluzione $\implies$ (U, F) ha una partizione. Supponiamo che esista un sottoinsieme di oggetti Knapsack S'_{items} tale che $\sum_{j \in S'_{items}} w_j = W$. Consideriamo la somma $\sum_{j \in S'_{items}} w_j$ in base $(M+1)$. Ogni w_j è un numero in base $(M+1)$ le cui cifre sono 0 o 1. La somma delle cifre in una data posizione k (corrispondente all'elemento $u_k \in U$) è data da $\sum_{j \in S'_{items}} \delta_{j,k}$. Poiché al massimo M numeri sono sommati (corrispondenti agli M insiemi in

F), e la base è $(M+1)$, non ci saranno mai riporti. Questo è cruciale: la somma di M cifre (0 o 1) in base $(M+1)$ non può generare un riporto. La somma massima possibile per una colonna è $M \times 1 = M$, che è una cifra valida in base $(M+1)$ (da 0 a M). Dato che la somma totale è $W = \sum_{k=1}^{N} 1 \cdot (M+1)^{N-k}$ (il numero con tutte le cifre 1 in base $M+1$) e non ci sono riporti, questo implica che per ogni posizione k, la somma delle cifre in quella posizione deve essere esattamente 1.

$$\forall k \in \{1, \ldots, N\}, \quad \sum_{j \in S'_{items}} \delta_{j,k} = 1$$

Questa condizione significa che ogni elemento $u_k \in U$ è contenuto in esattamente uno degli insiemi S_j il cui oggetto corrispondente è stato selezionato per la bisaccia. Pertanto, gli insiemi $\{S_j \mid j \in S'_{items}\}$ formano una partizione di U. Dunque, se l'istanza di Knapsack ha una soluzione, (U, F) ha una partizione. $\square$

3.4 Conclusione

Avendo dimostrato che Knapsack (decisione) è in NP e NP-hard (tramite riduzione da Exact Cover), concludiamo che **Knapsack è NP-completo**. Il problema Knapsack può essere formulato come un problema di **Programmazione Lineare Intera (ILP)**. Poiché Knapsack è NP-completo, e ILP è un problema più generale che include Knapsack come caso speciale, si deduce che la **Programmazione Lineare Intera è NP-hard**.