UNICAM
Università di Camerino
1336

Risoluzione di problemi di programmazione lineare intera binaria all'interno di un'applicazione web

Diego Ercoli
Relatore: Luca Tesei

Corso di Laurea Triennale in Informatica (Classe L-31), Scuola di Scienze e Tecnologie

Camerino, 10 Aprile 2018

Piano della presentazione

Generico problema di PLI con variabili decisionali binarie

Problema generale di massimizzazione, con vincoli di disuguaglianza lineari, espresso in maniera formale.

$$\max \quad c^T x$$

$$Ax \leq b$$

$$x_i \in \{0,1\}, \quad i = 1, \ldots, n$$

$A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$.

Branch and Bound

- Il problema originale viene suddiviso (Branching) in sottoproblemi più piccoli, partizionando l'insieme delle soluzioni ammissibili ($S_0 = \bigcup_{i=1}^{r} S_i$).
- Per ciascun i-esimo sottoproblema si risolve il suo rilassamento continuo (Bounding), che ha come regione ammissibile $T_i \supseteq S_i$. Lo scopo è quello di ottenere un limite superiore (**bound**).
- Ciascun sottoproblema viene chiuso se:
  - È stata trovata una soluzione intera; eventualmente si aggiorna la soluzione incombente.
  - Il bound è inferiore al valore della soluzione incombente.

Si ripete il procedimento scegliendo un sottoproblema aperto.

Branch and Bound

- Il problema originale viene suddiviso (Branching) in sottoproblemi più piccoli, partizionando l'insieme delle soluzioni ammissibili ($S_0 = \bigcup_{i=1}^{r} S_i$).
- Per ciascun i-esimo sottoproblema si risolve il suo rilassamento continuo (Bounding), che ha come regione ammissibile $T_i \supseteq S_i$. Lo scopo è quello di ottenere un limite superiore (**bound**).
- Ciascun sottoproblema viene chiuso se:
  - È stata trovata una soluzione intera; eventualmente si aggiorna la soluzione incombente.
  - Il bound è inferiore al valore della soluzione incombente.

Si ripete il procedimendo scegliendo un sottoproblema aperto.

Branch and Bound

- Il problema originale viene suddiviso (Branching) in sottoproblemi più piccoli, partizionando l'insieme delle soluzioni ammissibili ($S_0 = \bigcup_{i=1}^{r} S_i$).
- Per ciascun i-esimo sottoproblema si risolve il suo rilassamento continuo (Bounding), che ha come regione ammissibile $T_i \supseteq S_i$. Lo scopo è quello di ottenere un limite superiore (**bound**).
- Ciascun sottoproblema viene chiuso se:
  - È stata trovata una soluzione intera; eventualmente si aggiorna la soluzione incombente.
  - Il bound è inferiore al valore della soluzione incombente.

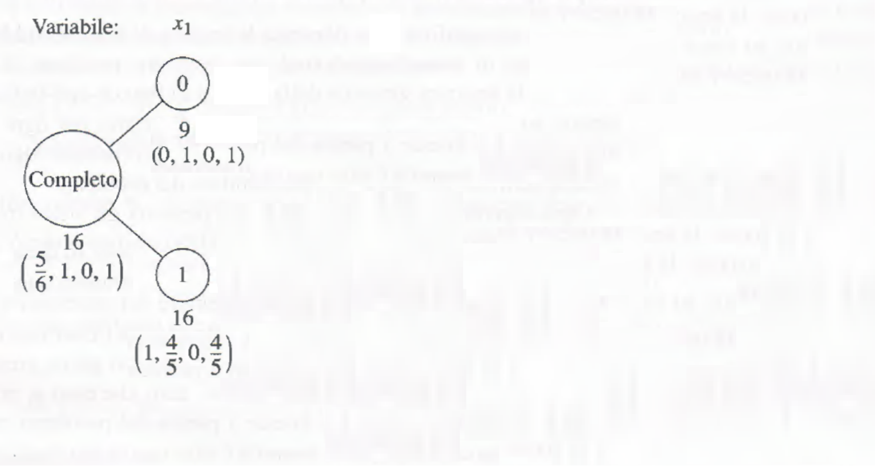
Si ripete il procedimendo scegliendo un sottoproblema aperto.

Branch and Bound

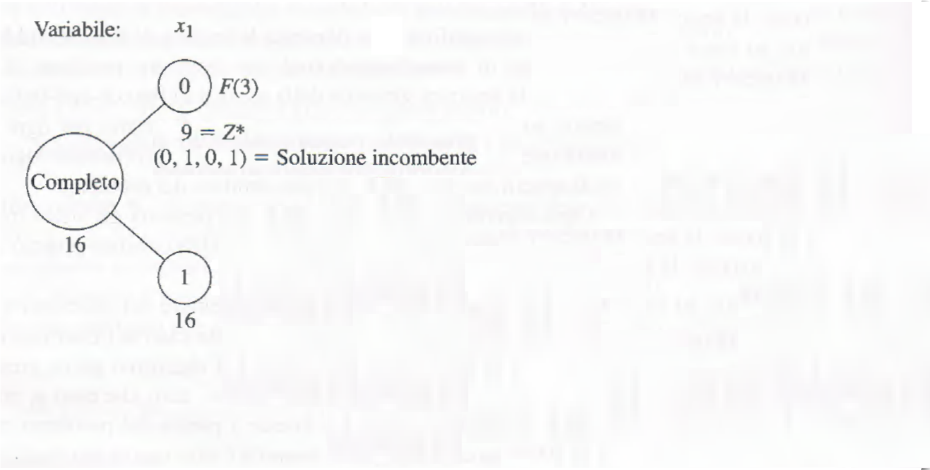
- Il problema originale viene suddiviso (Branching) in sottoproblemi più piccoli, partizionando l'insieme delle soluzioni ammissibili ($S_0 = \bigcup_{i=1}^{r} S_i$).
- Per ciascun i-esimo sottoproblema si risolve il suo rilassamento continuo (Bounding), che ha come regione ammissibile $T_i \supseteq S_i$. Lo scopo è quello di ottenere un limite superiore (**bound**).
- Ciascun sottoproblema viene chiuso se:
  - È stata trovata una soluzione intera; eventualmente si aggiorna la soluzione incombente.
  - Il bound è inferiore al valore della soluzione incombente.

Si ripete il procedimendo scegliendo un sottoproblema aperto.

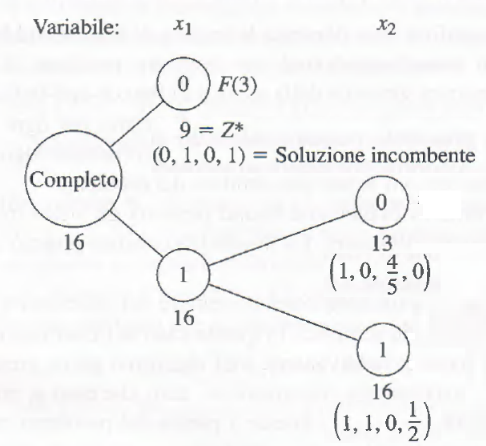
Branch and Bound applicato ad un problema di programmazione binaria



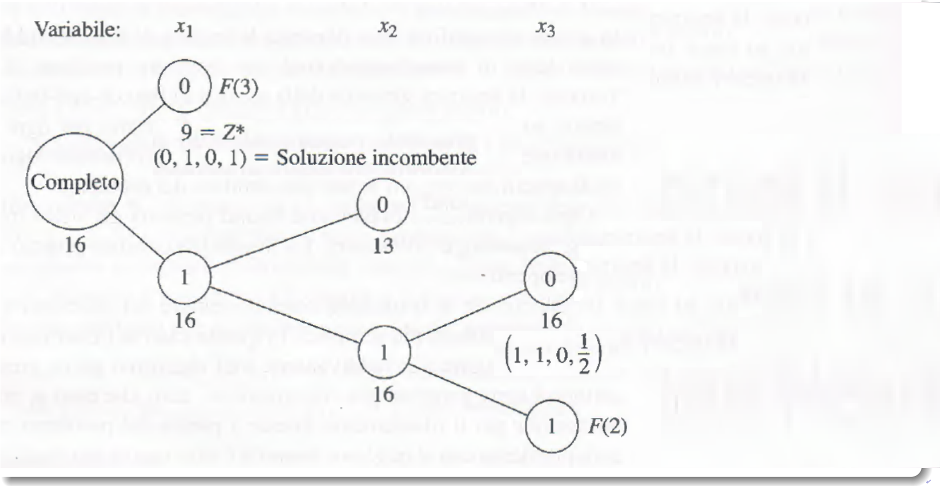
Branch and Bound applicato ad un problema di programmazione binaria



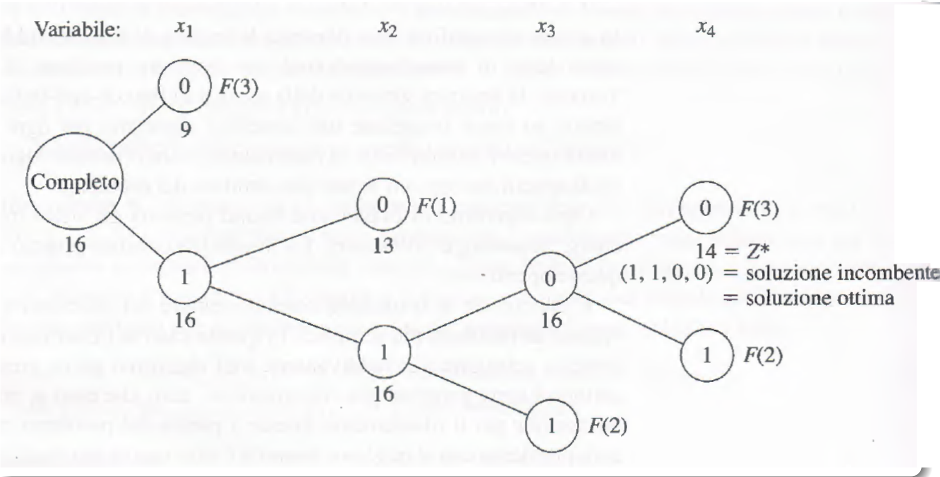
Branch and Bound applicato ad un problema di programmazione binaria



Branch and Bound applicato ad un problema di programmazione binaria



Branch and Bound applicato ad un problema di programmazione binaria



Principi Programmazione Dinamica

Requisiti:

- sottostruttura ottima: la soluzione ottima di un problema contiene al suo interno le soluzioni ottime di uno o più sottoproblemi correlati.
- sottoproblemi ripetuti: Il numero totale di sottoproblemi distinti dovrebbe essere un polinomio nella dimensione dell'input.

Si risolve uno specifico sottoproblema una sola volta, utilizzando i valori delle soluzioni ottime di appositi sottoproblemi di dimensione inferiore.
Si memorizza il valore appena calcolato.

Principi Algoritmo Goloso

Requisiti:

- proprietà della scelta golosa: Una soluzione globalmente ottima può essere sempre raggiunta facendo una **scelta localmente ottima**, dopo aver identificato un'opportuna strategia golosa.
- sottostruttura ottima: Combinando la scelta golosa con una soluzione ottima del sottoproblema generato, si ottiene una soluzione ottima del problema originale.

A differenza della programmazione dinamica:
per stabilire una scelta non c'è bisogno di risolvere particolari sottoproblemi.
Nota: in mancanza della proprietà della scelta golosa, in alcuni casi si può comunque ottenere un **algoritmo euristico** efficace.

Principi Algoritmo Goloso

Requisiti:

- proprietà della scelta golosa: Una soluzione globalmente ottima può essere sempre raggiunta facendo una **scelta localmente ottima**, dopo aver identificato un'opportuna strategia golosa.
- sottostruttura ottima: Combinando la scelta golosa con una soluzione ottima del sottoproblema generato, si ottiene una soluzione ottima del problema originale.

A differenza della programmazione dinamica:
per stabilire una scelta non c'è bisogno di risolvere particolari sottoproblemi.
Nota: in mancanza della proprietà della scelta golosa, in alcuni casi si può comunque ottenere un **algoritmo euristico** efficace.

Problema dello zaino 0-1

Formulazione matematica del problema.

$$\max \quad \sum_{i=1}^{n} v_i x_i$$

soggetto a:

$$\sum_{i=1}^{n} w_i x_i \leq W$$

$$x_i \in \{0,1\}, \quad i = 1, \dots, n$$

$v \in \mathbb{N}^n$, $w \in \mathbb{N}^n$, $W \in \mathbb{N}$.

Risoluzione con Branch and Bound

Il rilassamento continuo del problema consiste nel cancellare i vincoli di interezza. Si possono quindi selezionare frazioni degli oggetti.

Possiamo impiegare un semplice algoritmo goloso, consistente nel:

1. Ordinare gli elementi secondo il rapporto valore/peso.
2. Selezionare interamente gli elementi in sequenza, fino a che il costo del j-esimo elemento è inferiore alla capacità residua.
3. Selezionare una frazione del j-esimo elemento, pari al rapporto tra la capacità residua ed il suo costo.

Risoluzione con Branch and Bound

Il rilassamento continuo del problema consiste nel cancellare i vincoli di interezza. Si possono quindi selezionare frazioni degli oggetti.

Possiamo impiegare un semplice algoritmo goloso, consistente nel:

1. Ordinare gli elementi secondo il rapporto valore/peso.
2. Selezionare interamente gli elementi in sequenza, fino a che il costo del j-esimo elemento è inferiore alla capacità residua.
3. Selezionare una frazione del j-esimo elemento, pari al rapporto tra la capacità residua ed il suo costo.

Branch and Bound: complessità computazionale

Il tempo di esecuzione è teoricamente molto variabile; dipende da quanto è efficace la decomposizione del problema originario.

- **Caso pessimo:**
  verrebbe generato un albero binario completo con $\Theta(2^n)$ nodi.
  L'ordinamento viene eseguito solo all'inizio, la fase di bounding viene eseguita in ogni nodo con un tempo lineare $\Theta(n)$.
  Complessivamente $T(n)$ è $\Theta(n2^n)$.
- **Caso ottimo:**
  Viene individuata la soluzione ottima intera direttamente durante la fase di Bounding del nodo radice, quindi $T(n)$ è $\Theta(n \log n)$.

Generalizzando, il tempo di esecuzione è compreso tra $\Omega(n \log_2 n)$ e $\mathcal{O}(n2^n)$.

Branch and Bound: complessità computazionale

Il tempo di esecuzione è teoricamente molto variabile; dipende da quanto è efficace la decomposizione del problema originario.

- **Caso pessimo:**
  verrebbe generato un albero binario completo con $\Theta(2^n)$ nodi.
  L'ordinamento viene eseguito solo all'inizio, la fase di bounding viene eseguita in ogni nodo con un tempo lineare $\Theta(n)$.
  Complessivamente $T(n)$ è $\Theta(n2^n)$.
- **Caso ottimo:**
  Viene individuata la soluzione ottima intera direttamente durante la fase di Bounding del nodo radice, quindi $T(n)$ è $\Theta(n \log n)$.

Generalizzando, il tempo di esecuzione è compreso tra $\Omega(n \log_2 n)$ e $\mathcal{O}(n2^n)$.

Risoluzione con Programmazione Dinamica

- Spazio dei sottoproblemi:

$$P(I_{1\ldots\underline{n}},\, w),\ \text{con}\ 0 \leq \underline{n} \leq n\ \text{e}\ 0 \leq w \leq W$$

- Soluzione ricorsiva

$$V(\underline{n},\, w) = \begin{cases} 0 & \textbf{if } \underline{n} = 0 \textbf{ or } w = 0 \\ V(\underline{n}-1,\, w) & \textbf{if } w_{\underline{n}} > w \\ max\{V(\underline{n}-1,\, w) & \textbf{if } w_{\underline{n}} \leq w \\ \qquad v_{\underline{n}} + V(\underline{n}-1,\, w - w_{\underline{n}})\} & \end{cases} \tag{1}$$

- Complessivamente $T(n)$ è $\Theta(nW)$.

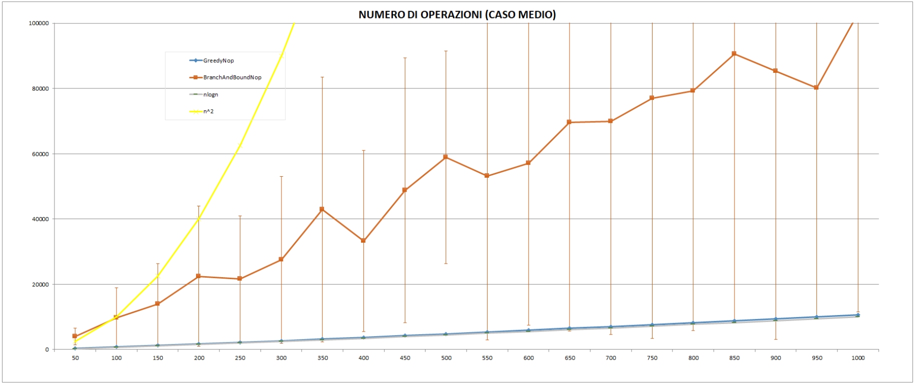
Algoritmo euristico

- Possiamo implementare un semplice algoritmo euristico, che usa la stessa strategia golosa descritta nel problema dello zaino "rilassato".
- La differenza fondamentale è che: se il costo dell'oggetto è inferiore alla capacità residua, allora viene scartato e si passa ad esaminare il successivo nell'ordinamento.
- L'algoritmo termina quando tutti gli oggetti sono stati esaminati oppure la capacià residua dello zaino diviene 0.
- $T(n)$ è $\Theta(n \log n)$.

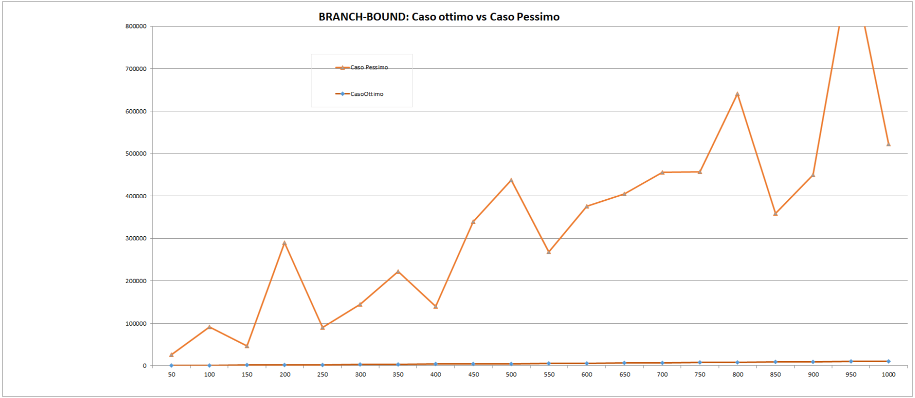
Algoritmo euristico

- Possiamo implementare un semplice algoritmo euristico, che usa la stessa strategia golosa descritta nel problema dello zaino "rilassato".
- La differenza fondamentale è che: se il costo dell'oggetto è inferiore alla capacità residua, allora viene scartato e si passa ad esaminare il successivo nell'ordinamento.
- L'algoritmo termina quando tutti gli oggetti sono stati esaminati oppure la capacià residua dello zaino diviene 0.
- $T(n)$ è $\Theta(n \log n)$.

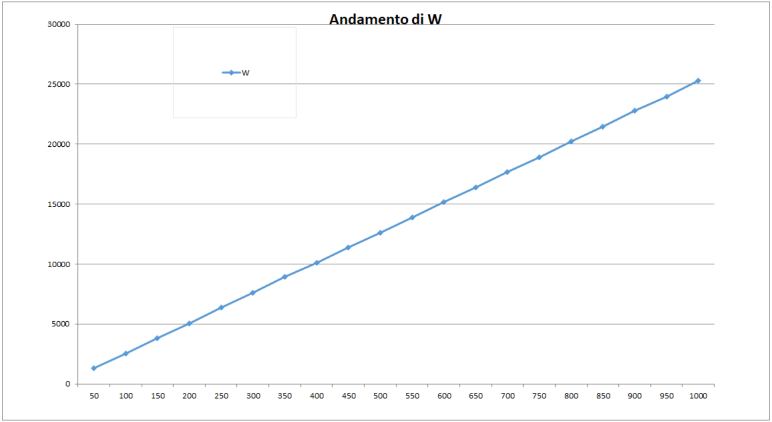
Benchmark: BranchBound vs Euristico



BranchBound: caso ottimo vs caso pessimo

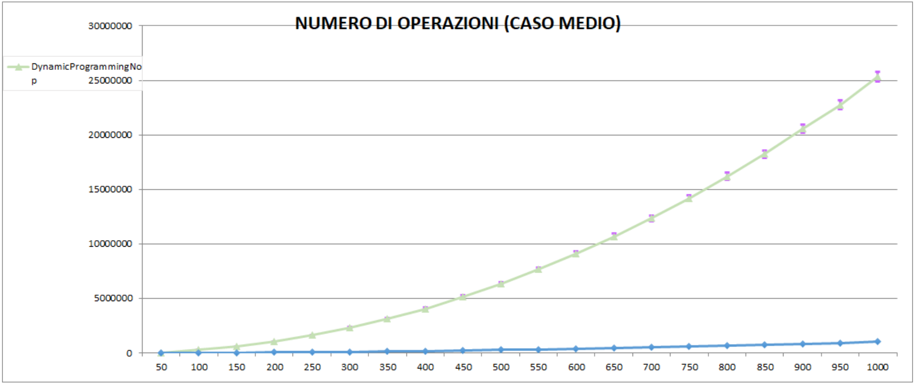


Capacità del problema in funzione di n

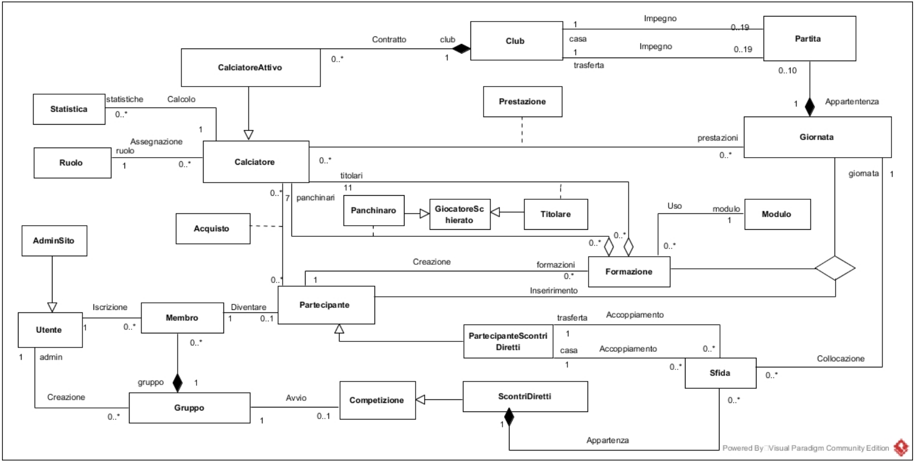


$$W = 0.5 \sum_{i=1}^{n} w_i.$$

ProgrammazioneDinamica

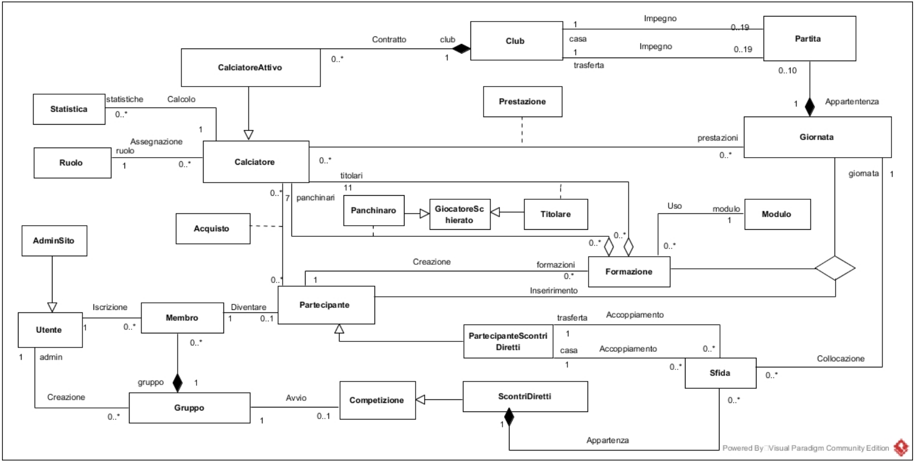


Modello concettuale del "fantacalcio"



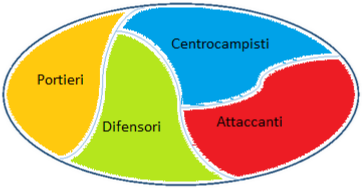
È stato utilizzato un servizio di ORM (Object-Relational Mapping).

Modello concettuale del "fantacalcio"



È stato utilizzato un servizio di ORM (Object-Relational Mapping).

Descrizione del problema dell'acquisto dei calciatori



- I sottoinsiemi P (Portieri), D (Difensori), C (Centrompasti), A (Attaccanti) costituiscono una partizione dell'insieme universo dei calciatori.
- Ogni i-esimo calciatore ha un costo w_i ed un valore v_i (media dei voti).
- Per costruire la propria rosa si devono comprare esattamente un certo numero di calciatori per ruolo.

Definizione formale del problema dell'acquisto dei calciatori

$$\max \quad \sum_{i=1}^{n} v_i x_i$$

$$\sum_{i=1}^{n} w_i x_i \leq budget$$

$$x_i \in \{0,1\}, \quad i \in N = \{1, \dots, n\}$$

Vincoli aggiuntivi:

$$\sum_{i \in N_r} x_i = cont_r, \; r = 1, \dots, m$$

$$N = \bigcup_{r=1}^{m} N_r$$

$$\forall h, k \in \{1, \dots, m\} \, (h \neq k \implies N_h \cap N_k = \emptyset)$$

Definizione formale del problema dell'acquisto dei calciatori

$$\max \quad \sum_{i=1}^{n} v_i x_i$$

$$\sum_{i=1}^{n} w_i x_i \leq budget$$

$$x_i \in \{0,1\}, \quad i \in N = \{1,\ldots,n\}$$

Vincoli aggiuntivi:

$$\sum_{i \in N_r} x_i \;= cont_r,\; r = 1,\ldots,m$$

$$N = \bigcup_{r=1}^{m} N_r$$

$$\forall h,k \in \{1,\ldots,m\}\,(h \neq k \implies N_h \cap N_k = \emptyset)$$

Risoluzione con Branch and Bound

- Nella fase di Bounding il rilassamento continuo del problema non può essere risolto con un algoritmo goloso; dobbiamo utilizzare un algoritmo più costoso.
- Impieghiamo l'algoritmo del simplesso ampiamente utilizzato in Ricerca Operativa per risolvere problemi di programmazione lineare.
- Dobbiamo fornire in input il modello matematico del problema in questione:
  - il tipo di funzione obiettivo (massimizzazione/minimizzazione);
  - il vettore $c \in \mathbb{R}^n$ dei coefficenti della funzione obiettivo;
  - la matrice $A \in \mathbb{R}^{m \times n}$ dei coefficenti dei vincoli;
  - il vettore $b \in \mathbb{R}^m$ dei coefficenti dei termini noti.

Risoluzione con Branch and Bound

- Nella fase di Bounding il rilassamento continuo del problema non può essere risolto con un algoritmo goloso; dobbiamo utilizzare un algoritmo più costoso.
- Impieghiamo l'algoritmo del simplesso ampiamente utilizzato in Ricerca Operativa per risolvere problemi di programmazione lineare.
- Dobbiamo fornire in input il modello matematico del problema in questione:
  - il tipo di funzione obiettivo (massimizzazione/minimizzazione);
  - il vettore $c \in \mathbb{R}^n$ dei coefficenti della funzione obiettivo;
  - la matrice $A \in \mathbb{R}^{m \times n}$ dei coefficenti dei vincoli;
  - il vettore $b \in \mathbb{R}^m$ dei coefficenti dei termini noti.

Risultato risoluzione con Branch and Bound

- Il numero totale di combinazioni dei calciatori è davvero elevato (incluse quelle non ammissibili):

$$\binom{20}{3} * \binom{91}{8} * \binom{99}{8} * \binom{99}{8}$$

- L'algoritmo ha impiegato meno di un minuto per trovare la soluzione ottima.

- Verifichiamo se mediante la Programmazione Dinamica, riusciamo a progettare un procedimento risolutivo più efficiente.

Risultato risoluzione con Branch and Bound

- Il numero totale di combinazioni dei calciatori è davvero elevato (incluse quelle non ammissibili):

$$\binom{20}{3} * \binom{91}{8} * \binom{99}{8} * \binom{99}{8}$$

- L'algoritmo ha impiegato meno di un minuto per trovare la soluzione ottima.

- Verifichiamo se mediante la Programmazione Dinamica, riusciamo a progettare un procedimento risolutivo più efficiente.

Programmazione Dinamica: Generico sottoproblema

- Sia $I = \{\ I_i \mid i \in N\}$ l'insieme universo dei calciatori acquistabili.

- Ogni sottoinsieme $N_r \subset N$ contiene un specifico insieme di indici associati a I,ovvero $N_r = \{j_1^r, \dots, j_{n_r}^r\}$.
  Denotiamo la lista dei calciatori di ruolo r con $I^r = \{I_{j_1^r}, \dots, I_{j_{n_r}^r}\}$.

- Sia $P_r(cont, i, w)$ il generico sottoproblema in cui occorre esaminare r sottoinsiemi e nel sottoinsieme r-esimo (quello corrente) è necessario selezionare $cont$ tra quelli i rimanenti.

Programmazione Dinamica: Generico sottoproblema

- Sia $I = \{\ I_i \mid i \in N\}$ l'insieme universo dei calciatori acquistabili.
- Ogni sottoinsieme $N_r \subset N$ contiene un specifico insieme di indici associati a I,ovvero $N_r = \{j_1^r, \dots, j_{n_r}^r\}$.
  Denotiamo la lista dei calciatori di ruolo r con $I^r = \{I_{j_1^r}, \dots, I_{j_{n_r}^r}\}$.
- Sia $P_r(cont, i, w)$ il generico sottoproblema in cui occorre esaminare r sottoinsiemi e nel sottoinsieme r-esimo (quello corrente) è necessario selezionare *cont* tra quelli i rimanenti.

Programmazione Dinamica: Spazio dei sottoproblemi

- Dato $P_r(cont, i, w)$, prendendo come riferimento l'ultimo elemento $I_{j_i^r}$ potremmo:
  1. Scegliere l'elemento e risolvere il problema $P_r(cont-1, i-1, w-w_{j_i^r})$
  2. Ignorare l'elemento e risolvere il problema $P_r(cont, i-1, w)$ (In caso $w < w_{j_i^r}$ viene fatta questa scelta).

  Viene effettuata la scelta più conveniente.

- Casi particolari:
  - se $i < cont$: non vi sono soluzioni ammissibili;
  - se $cont = 0$: si esamina $P_{r-1}(cont_{r-1}, n_{r-1}, w)$;
  - se $r = 0$: **caso base**

- Spazio dei sottoproblemi:
  $\{ P_r(cont, i, w) \mid 0 \le r \le m, 1 \le 1 \le n_r, 0 \le w \le W, 0 \le cont \le cont_r \}$

Programmazione Dinamica: Spazio dei sottoproblemi

- Dato $P_r(cont, i, w)$, prendendo come riferimento l'ultimo elemento $l_{j_i^r}$ potremmo:
  1. Scegliere l'elemento e risolvere il problema $P_r(cont-1, i-1, w-w_{j_i^r})$
  2. Ignorare l'elemento e risolvere il problema $P_r(cont, i-1, w)$ (In caso $w < w_{j_i^r}$ viene fatta questa scelta).

  Viene effettuata la scelta più conveniente.

- Casi particolari:
  - se $i < cont$: non vi sono soluzioni ammissibili;
  - se $cont = 0$: si esamina $P_{r-1}(cont_{r-1}, n_{r-1}, w)$;
  - se $r = 0$: **caso base**

- Spazio dei sottoproblemi:
  $\{ P_r(cont, i, w) \mid 0 \leq r \leq m, 1 \leq 1 \leq n_r, 0 \leq w \leq W, 0 \leq cont \leq cont_r \}$

Programmazione Dinamica: Spazio dei sottoproblemi

- Dato $P_r(cont, i, w)$, prendendo come riferimento l'ultimo elemento $I_{j_i^r}$ potremmo:
  1. Scegliere l'elemento e risolvere il problema $P_r(cont-1, i-1, w-w_{j_i^r})$
  2. Ignorare l'elemento e risolvere il problema $P_r(cont, i-1, w)$ (In caso $w < w_{j_i^r}$ viene fatta questa scelta).

  Viene effettuata la scelta più conveniente.

- Casi particolari:
  - se $i < cont$: non vi sono soluzioni ammissibili;
  - se $cont = 0$: si esamina $P_{r-1}(cont_{r-1}, n_{r-1}, w)$;
  - se $r = 0$: **caso base**

- Spazio dei sottoproblemi:
  $\{ P_r(cont, i, w) \mid 0 \leq r \leq m, 1 \leq 1 \leq n_r, 0 \leq w \leq W, 0 \leq cont \leq cont_r \}$

Programmazione Dinamica: Spazio dei sottoproblemi

- Dato $P_r(cont, i, w)$, prendendo come riferimento l'ultimo elemento $I_{j_i^r}$ potremmo:
  1. Scegliere l'elemento e risolvere il problema $P_r(cont-1, i-1, w-w_{j_i^r})$
  2. Ignorare l'elemento e risolvere il problema $P_r(cont, i-1, w)$ (In caso $w < w_{j_i^r}$ viene fatta questa scelta).

  Viene effettuata la scelta più conveniente.

- Casi particolari:
  - se $i < cont$: non vi sono soluzioni ammissibili;
  - se $cont = 0$: si esamina $P_{r-1}(cont_{r-1}, n_{r-1}, w)$;
  - se $r = 0$: **caso base**

- Spazio dei sottoproblemi:
  $\{ P_r(cont, i, w) \mid 0 \le r \le m, 1 \le 1 \le n_r, 0 \le w \le W, 0 \le cont \le cont_r \}$

Programmazione Dinamica: Spazio dei sottoproblemi

- Dato $P_r(cont, i, w)$, prendendo come riferimento l'ultimo elemento $I_{j_i^r}$ potremmo:
  1. Scegliere l'elemento e risolvere il problema $P_r(cont-1, i-1, w-w_{j_i^r})$
  2. Ignorare l'elemento e risolvere il problema $P_r(cont, i-1, w)$ (In caso $w < w_{j_i^r}$ viene fatta questa scelta).

  Viene effettuata la scelta più conveniente.

- Casi particolari:
  - se $i < cont$: non vi sono soluzioni ammissibili;
  - se $cont = 0$: si esamina $P_{r-1}(cont_{r-1}, n_{r-1}, w)$;
  - se $r = 0$: **caso base**

- Spazio dei sottoproblemi:
  $\{ P_r(cont, i, w) \mid 0 \leq r \leq m, 1 \leq 1 \leq n_r, 0 \leq w \leq W, 0 \leq cont \leq cont_r \}$

Programmazione Dinamica: Sottostruttura Ottima

- Sia $a = \{I_{j_i^r}\} \cup a'$ la soluzione ottima di $P_r(cont, i-1, w)$; dimostriamo che a' è la soluzione ottima di $P_r(cont-1, i-1, w-w_{j_i^r})$.
- Per assurdo esiste a'' con valore $v(a'')$ maggiore di a'.
- Sostituitamo a' con a'' in a, generando una nuova soluzione $\bar{a}$.
- Il fatto che $V(\bar{a}) > V(a)$ **contraddice** l'ipotesi che a sia una soluzione ottima.

Se $\{I_{j_i^r}\} \notin a$, con una dimostrazione analoga si può dimostrare che a è la soluzione ottima di $P_r(cont, i-1, w)$.

Programmazione Dinamica: Sottostruttura Ottima

- Sia $a = \{I_{j_i^r}\} \cup a'$ la soluzione ottima di $P_r(cont, i-1, w)$; dimostriamo che a' è la soluzione ottima di $P_r(cont-1, i-1, w-w_{j_i^r})$.
- Per assurdo esiste a'' con valore $v(a'')$ maggiore di a'.
- Sostituitamo a' con a'' in a, generando una nuova soluzione $\bar{a}$.
- Il fatto che $V(\bar{a}) > V(a)$ **contraddice** l'ipotesi che a sia una soluzione ottima.

Se $\{I_{j_i^r}\} \notin a$, con una dimostrazione analoga si può dimostrare che a è la soluzione ottima di $P_r(cont, i-1, w)$.

Programmazione Dinamica: Sottostruttura Ottima

- Sia $a = \{I_{j_i^r}\} \cup a'$ la soluzione ottima di $P_r(cont, i-1, w)$; dimostriamo che a' è la soluzione ottima di $P_r(cont-1, i-1, w-w_{j_i^r})$.
- Per assurdo esiste a'' con valore $v(a'')$ maggiore di a'.
- Sostituitamo a' con a'' in a, generando una nuova soluzione $\bar{a}$.
- Il fatto che $V(\bar{a}) > V(a)$ **contraddice** l'ipotesi che a sia una soluzione ottima.

Se $\{I_{j_i^r}\} \notin a$, con una dimostrazione analoga si può dimostrare che a è la soluzione ottima di $P_r(cont, i-1, w)$.

Programmazione Dinamica: Soluzione ricorsiva

$$V_r(cont,\, i,\, w) = \begin{cases} 0 & \textbf{if } r = 0 \\ V_{r-1}(cont = cont_{r-1},\, i = n_{r-1},\, w) & \textbf{else if } cont = 0 \\ -\infty & \textbf{else if } i < cont \\ V_r(cont,\, i-1,\, w) & \textbf{else if } w < w_{j_i^r} \\ max\{V_r(cont,\, i-1,\, w), & \textbf{else} \\ \ v_{j_i^r} + V_r(cont-1,\, i-1,\, w - w_{j_i^r})\} & \end{cases} \tag{2}$$

Programmazione Dinamica: Numero di sottoproblemi distinti

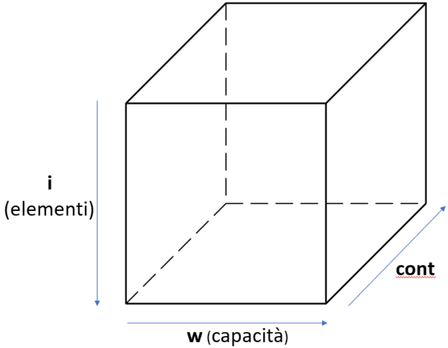
- Un algoritmo ricorsivo basato su tale ricorrenza risulterebbe esponenziale, generando un albero binario di ricorsione non completo con $\mathcal{O}(2^n)$ nodi.
- Tuttavia i sottoproblemi si ripetono, infatti il numero totale di sottoproblemi distinti non è esponenziale:

$$\sum_{r=1}^{m} cont_r n_r W = \mathcal{O}(cont_{max} n W)$$

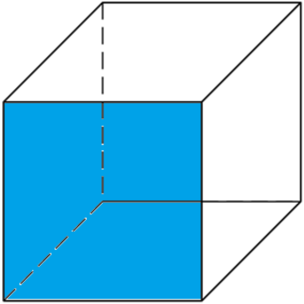
$$\text{con } n = \sum_{r=1}^{m} n_r \quad \text{e} \quad cont_{max} = \max_{1 \leq r \leq m} \{cont_r\}$$

Programmazione Dinamica: approccio bottom-up

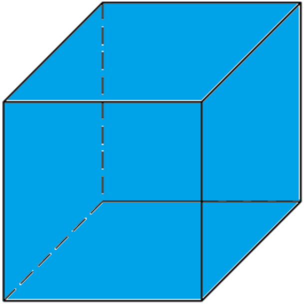
Per ogni sottoinsieme r, allochiamo un **array tridimensionale** di questo tipo:



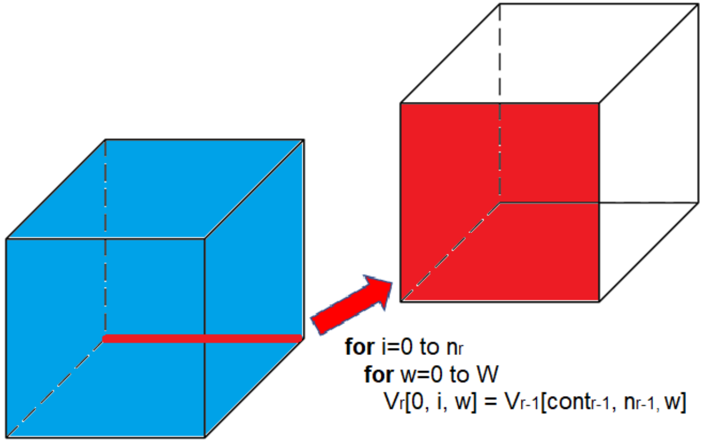
Programmazione Dinamica: approccio bottom-up



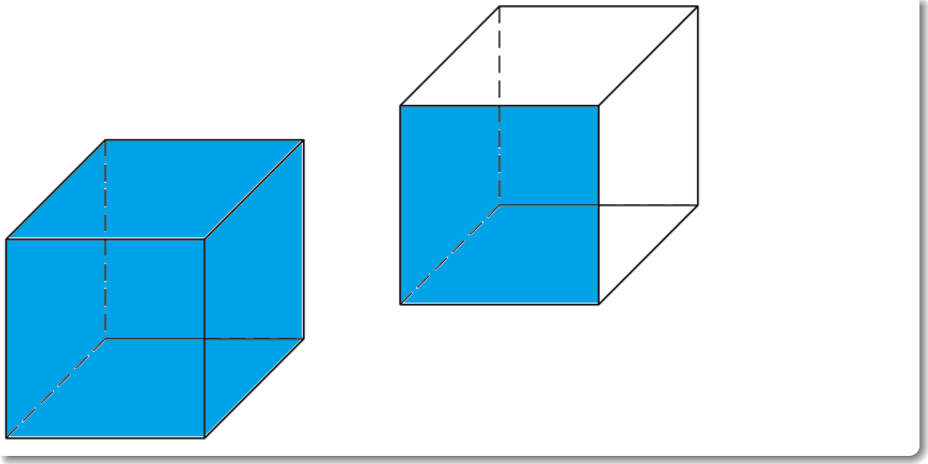
Programmazione Dinamica: approccio bottom-up



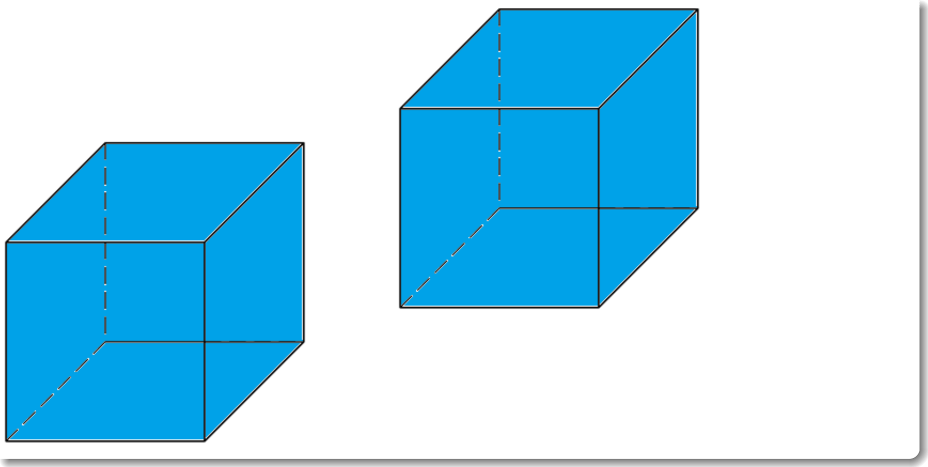
Programmazione Dinamica: approccio bottom-up



Programmazione Dinamica: approccio bottom-up



Programmazione Dinamica: approccio bottom-up



Descrizione del problema della formazione

Data una rosa di 25 calciatori, si vuole schierare la miglior formazione possibile scegliendo 11 calciatori e rispettando i seguenti vincoli:

- in ogni formazione ci deve essere un portiere;
- è possibile utilizzare uno dei seguenti moduli (D-C-A): 3-4-3, 3-5-2, 4-3-3, 4-4-2, 5-3-2.

Definizione formale del problema della formazione

$$\max \sum_{i \in ROSA} v_i x_i$$

$$\sum_{i \in ROSA} x_i = 11$$

$$\sum_{i \in P} x_i = 1 \quad 3 \leq \sum_{i \in D} x_i \leq 5$$

$$3 \leq \sum_{i \in C} x_i \leq 5 \quad 1 \leq \sum_{i \in A} x_i \leq 3$$

$$x_i \in \{0, 1\}, \quad i \in ROSA = \{1, \ldots, 25\}$$

$$P \cup D \cup C \cup A = ROSA$$

$$P \cap D = \emptyset,\ P \cap C = \emptyset,\ P \cap A = \emptyset,\ D \cap C = \emptyset,\ D \cap A = \emptyset,\ C \cap A = \emptyset$$

Algoritmo goloso: Strategia golosa

- Denotiamo il problema con $P(m, min, max, titolari)$.

- Ad ogni iterazione viene esaminato il calciatore di maggior valore tra quelli rimasti, sia r il suo ruolo.

- Il calciatore può essere scelto se almeno una delle due condizioni è vera, in tal caso si decrementano min_r e max_r e si risolve il sottoproblema $P(n-1, min, max, titolari - 1)$.
  1. se $min_r > 0$
  2. se $(tot - \sum_i min_i) > 0$ e $min_r > 0$

Altrimenti si risolve il sottoproblema $P(n-1, min, max, titolari)$.

Algoritmo goloso: Strategia golosa

- Denotiamo il problema con $P(m, min, max, titolari)$.

- Ad ogni iterazione viene esaminato il calciatore di maggior valore tra quelli rimasti, sia r il suo ruolo.

- Il calciatore può essere scelto se almeno una delle due condizioni è vera, in tal caso si decrementano min_r e max_r e si risolve il sottoproblema $P(n-1, min, max, titolari-1)$.
  1. se $min_r > 0$
  2. se $(tot - \sum_i min_i) > 0$ e $min_r > 0$

Altrimenti si risolve il sottoproblema $P(n-1, min, max, titolari)$.

Demo del software