

## Software testing.

Una software house deve controllare la correttezza di un componente software critico nell'ambito di una missione spaziale. A questo scopo vuole eseguire il software in diverse configurazioni. Tuttavia, essa non ha tempo per testare tutte le possibili configurazioni, poiché esse corrispondono a tutte le  $2^n$  possibili combinazioni di un dato numero  $n$  di parametri binari.

Gli ingegneri del software si accontentano quindi di includere nel loro programma di test un sottoinsieme di tutte le possibili combinazioni, che abbia la caratteristica di contenere tutte le possibili  $2^t$  combinazioni per ogni sottoinsieme di  $t$  parametri.

L'obiettivo è quello di minimizzare il numero di test necessari, cioè di configurazioni da testare.

Formulare il problema e classificarlo. Risolvere l'esempio con  $p = 5$  e  $t = 3$ , discutendo ottimalità e unicità della soluzione trovata.

Per maggiore chiarezza, si riporta qui di seguito la soluzione ottima per  $n = 4$  parametri e  $t = 2$ . Essa consiste nelle 5 configurazioni di test indicate in tabella, che sono composte in modo tale che tutte le coppie di valori binari possibili (cioè 00, 01, 10 e 11) compaiono su ogni coppia di colonne.

0	0	0	0
1	1	1	0
1	1	0	1
1	0	1	1
0	1	1	1

## Soluzione.

Il problema è noto come Covering Array Problem ed è un problema reale in ingegneria del software. Si può formulare in termini di Programmazione Lineare Intera come segue.

### Dati.

Sia  $n$  il numero di parametri e sia  $N$  il loro insieme indicizzato. Sia  $c$  il numero delle configurazioni utilizzabili (stima per eccesso) e sia  $C$  il loro insieme indicizzato. Un limite superiore al valore di  $c$  è naturalmente  $2^n$ . Se si suppone di poter usare una matrice binaria di dati con dimensione  $n \times 2^n$ , allora il problema si riduce a selezionare alcune delle  $2^n$  combinazioni tra tutte quelle possibili. Nello svolgimento dell'esercizio, dovendo risolvere solo un esempio piccolo, il problema dell'esplosione combinatoria non si pone. Tuttavia, l'obiettivo dell'esercizio è quello di contrastare l'esplosione combinatoria in  $n$ , evitando quindi di utilizzare dati di dimensione esponenziale in  $n$ . Meglio, quindi, stimare un upper bound al valore di  $c$  ed eventualmente aumentarlo se il problema risulta inammissibile, oppure stimare la quantità di memoria utilizzabile sull'hardware a disposizione e dimensionare di conseguenza l'upper bound. La difficoltà, in questo caso, sta nel fatto che la matrice delle configurazioni non è data, ma è variabile.

Sia  $t$  la cardinalità dei sottoinsiemi da testare e sia  $T$  l'insieme dei loro indici da 0 a  $2^t - 1$ . Nell'esempio proposto, con  $t = 3$ , sia hanno 8 possibili configurazioni, da 000 con indice 0 a 111 con indice 7.

### Variabili.

Le variabili del problema devono descrivere le  $c$  configurazioni usate per fare il test. Ogni configurazione è un vettore binario con  $n$  componenti. Quindi le variabili possono essere descritte da una matrice binaria  $x$  con  $c$  righe e  $n$  colonne.

Per indicare se una data configurazione copre o no una delle terne su un terna di colonne, si possono usare altre variabili binarie  $y$  con cinque indici: i primi tre indici  $i, j$  e  $k$  (compresi tra 1 e  $n$ , ordinati e diversi tra loro) determinano le tre colonne, il quarto indice  $m$  (tra 0 e  $2^t - 1$ ) identifica la tupla (considerandola come un numero intero in base 2) ed il quinto indice  $r$  (da 1 a  $c$ ) identifica la configurazione di test. La variabile  $y_{ijkmr}$  vale 1 se e solo se nella configurazione  $r$  compare la tripla  $m$ -esima sulle colonne  $i, j$  e  $k$ .

### Vincoli.

Per mettere in relazione le variabili  $x$  e  $y$  si possono usare i seguenti vincoli. Per la prima colonna:

$$x_{ri} \leq ((m \div 4) \bmod 2) + (1 - y_{ijkmr}) \quad \forall i \in N, j \in N, k \in N, m \in T, r \in C : (j > i) \wedge (k > j)$$

$$x_{ri} \geq ((m \div 4) \bmod 2) - (1 - y_{ijkmr}) \quad \forall i \in N, j \in N, k \in N, m \in T, r \in C : (j > i) \wedge (k > j)$$

Per la seconda colonna:

$$x_{rj} \leq ((m \div 2) \bmod 2) + (1 - y_{ijkmr}) \quad \forall i \in N, j \in N, k \in N, m \in T, r \in C : (j > i) \wedge (k > j)$$

$$x_{rj} \geq ((m \div 2) \bmod 2) - (1 - y_{ijkmr}) \quad \forall i \in N, j \in N, k \in N, m \in T, r \in C : (j > i) \wedge (k > j)$$

Per la terza colonna:

$$x_{rk} \leq (m \bmod 2) + (1 - y_{ijkmr}) \quad \forall i \in N, j \in N, k \in N, m \in T, r \in C : (j > i) \wedge (k > j)$$

$$x_{rk} \geq (m \bmod 2) - (1 - y_{ijkmr}) \quad \forall i \in N, j \in N, k \in N, m \in T, r \in C : (j > i) \wedge (k > j)$$

Questi vincoli forzano una uguaglianza tramite due disuguaglianze di segno opposto quando la variabile  $y$  vale 1 e sono inattivi quando la variabile  $y$  vale 0. Così  $y_{ijkmr} = 1$  forza  $x_{ri} = ((m \div 4) \bmod 2)$ ,  $x_{rj} = ((m \div 2) \bmod 2)$  e  $x_{rk} = (m \bmod 2)$ , cioè forza la riga  $r$  della matrice  $x$  a contenere sulle colonne  $i, j$  e  $k$  i valori binari della terna di indice  $m$ . Il valore  $((m \div 4) \bmod 2)$  è pari a 0 per  $m = 0, 1, 2, 3$  e 1 per  $m = 4, 5, 6, 7$ : è il bit più significativo della terna di indice  $m$ ; il valore  $((m \div 2) \bmod 2)$  è pari a 0 per  $m = 0, 1, 4, 5$  e 1 per  $m = 2, 3, 6, 7$ : è il bit mediano della terna di indice  $m$ ; il valore  $(m \bmod 2)$  è pari a 0 per  $m = 0, 2, 4, 6$  e 1 per  $m = 1, 3, 5, 7$ : è il bit meno significativo della terna di indice  $m$ .

Un modo più sintetico di ottenere lo stesso effetto sfrutta l'unicità della codifica binaria, grazie alla quale la tupla formata dai valori sulle tre colonne corrisponde alla tupla  $m$ -esima se e solo se corrisponde allo stesso numero intero espresso in base 2.

$$4x_{ri} + 2x_{rj} + x_{rk} \leq m + 8(1 - y_{ijkmr}) \quad \forall i \in N, j \in N, k \in N, m \in T, r \in C : (j > i) \wedge (k > j)$$

$$4x_{ri} + 2x_{rj} + x_{rk} \geq m - 8(1 - y_{ijkmr}) \quad \forall i \in N, j \in N, k \in N, m \in T, r \in C : (j > i) \wedge (k > j)$$

Il vincolo sulla presenza nel test di tutte le terne per tutte le terne di colonne è:

$$\sum_{r \in C} y_{ijkmr} \geq 1 \quad \forall i \in N, j \in N, k \in N, m \in T : (j > i) \wedge (k > j).$$

Per formulare l'obiettivo di minimizzare il numero di configurazioni usate, servono variabili binarie  $w_r$  per ogni possibile configurazione  $r \in C$ , che indicano se la configurazione è utilizzata o no.

Tali variabili compaiono nei vincoli

$$y_{ijkmr} \leq w_r \quad \forall i \in N, j \in N, k \in N, m \in T, r \in C : (j > i) \wedge (k > j)$$

che forzano ad 1 la variabile  $w_r$  quando almeno una terna viene testata sulla riga  $r$ .

Per aiutare il solutore a convergere, può essere utile inserire dei vincoli che rompano la simmetria, come ad esempio

$$w_r \leq w_{t-1} \quad \forall r \in C : r > 1$$

o, ancora più forti,

$$\sum_{k \in N} 2^{(k-1)} x_{rk} \leq \sum_{k \in N} 2^{(k-1)} x_{r-1,k} \quad \forall r \in C : r > 1.$$

*Obiettivo.*

L'obiettivo è la minimizzazione del numero di configurazioni utilizzate:

$$\text{minimize } c = \sum_{r \in C} w_r.$$

*Classificazione.*

Il modello risultante è di PLI. L'ottimalità della soluzione prodotta dai solutori è garantita; l'unicità no. È possibile anche formulare il problema con un modello di PNL, evitando i vincoli lineari disattivabili per collegare le variabili  $x$  e le variabili  $y$ .

Nell'esempio con  $n = 5$  e  $t = 3$  il minimo numero di configurazioni necessarie è 10, come indicato nella tabella seguente.

	1	2	3	4	5
1	1	1	1	1	1
2	0	0	1	1	1
3	0	1	0	1	1
4	0	1	1	0	1
5	1	0	0	0	1
6	0	1	1	1	0
7	1	0	0	1	0
8	1	0	1	0	0
9	1	1	0	0	0
10	0	0	0	0	0