

Colorazione dei vertici: il branch-and-bound DSATUR

Indice

L'algoritmo per il VCP

Branching

Bound e potatura

Scelta del nodo e pseudocodice

Correttezza

Un esempio svolto

Osservazioni

Errori tipici

In sintesi

Obiettivi

Al termine di questo capitolo saremo in grado di:

- ▶ descrivere un nodo dell'albero come colorazione parziale, con i suoi colori usati e i suoi gradi di saturazione;
- ▶ fare branching sul vertice di grado di saturazione massimo, con un figlio per ogni colore ammissibile e uno per un colore nuovo;
- ▶ calcolare l'upper bound con *DSATUR* e il lower bound con una clique greedy, e potare un nodo;
- ▶ eseguire a mano l'algoritmo di branch-and-bound *DSATUR* e dimostrare che è esatto.

Prerequisiti

- ▶ Dal modulo *Programmazione lineare intera*, il capitolo «[Branch-and-bound](#)»: operazioni di branching e di bounding, lista dei problemi, criteri di potatura, bound LB e UB.
- ▶ Da questo modulo, il capitolo «[Problemi di colorazione](#)»: il problema della colorazione dei vertici, clique, insiemi stabili e il bound della clique $\omega(G) \leq \chi(G)$.
- ▶ Da questo modulo, il capitolo «[Euristiche costruttive](#)»: il grado di saturazione e l'euristica *DSATUR*.

Il problema di questo capitolo

Il problema di questo capitolo è il
problema della colorazione dei vertici (VCP):

la sua definizione e le sue ipotesi sono nella sua pagina.

- ▶ Su un grafo $G = (V, E)$, il suo valore ottimo è il numero cromatico $\chi(G)$
- ▶ Scriviamo un algoritmo esatto per il VCP: un algoritmo di branch-and-bound i cui nodi sono colorazioni parziali, e i cui bound vengono dal conteggio dei colori

L'algoritmo per il VCP

Introduzione: L'algoritmo per il VCP

- ▶ In questa sezione costruiamo l'algoritmo di branch-and-bound DSATUR, un pezzo alla volta: i nodi dell'albero, la regola di branching, i bound e la potatura, la scelta del nodo
- ▶ La regola di branching è la regola dell'euristica *DSATUR*: colorare per primo il vertice più vincolato
- ▶ Alla fine dimostriamo che l'algoritmo è esatto, e che il suo albero non contiene copie simmetriche della stessa colorazione

I nodi dell'albero

- ▶ In un algoritmo di branch-and-bound per il VCP, un nodo dell'albero deve descrivere un insieme di colorazioni
- ▶ Qui un nodo è una **colorazione parziale**: alcuni vertici hanno già un colore, gli altri no
- ▶ Definiamo prima una colorazione parziale e le quantità che calcoliamo in un nodo

Nodo di branching e colorazione parziale (1/2)

Definizione 1: Nodo di branching e colorazione parziale

*Nell'algoritmo di branch-and-bound DSATUR i colori sono gli interi $1, 2, 3, \dots$, e un nodo di branching k dell'albero è una **colorazione parziale**:*

1. *un sottoinsieme di **vertici colorati** $\rightsquigarrow$ indicato con $V^k \subseteq V$;*
2. *un colore $col^k(v)$ per ogni $v \in V^k$, con $col^k(u) \neq col^k(v)$ per ogni $\{u, v\} \in E$, $u, v \in V^k$;*
3. *un sottoinsieme di **vertici non colorati** $\rightsquigarrow$ indicato con $U^k := V \setminus V^k$.*

*I colori usati nel nodo k sono $1, 2, \dots, q(k)$, dove $q(k)$ è il **numero di colori usati**, e le classi di colore sono $Q_c^k := \{v \in V^k : col^k(v) = c\}$, $c = 1, \dots, q(k)$.*

Nodo di branching e colorazione parziale (2/2)

Definizione: Nodo di branching e colorazione parziale (continua)

Per un vertice non colorato $v \in U^k$:

- *il grado di saturazione*

$$\text{sat}_k(v) := |\{ \text{col}^k(u) : u \in N(v) \cap V^k \}|;$$

- *il grado fra i vertici non colorati*

$$du_k(v) := |N(v) \cap U^k|;$$

- *i colori ammissibili*

$$F_k(v) := \{ c \in \{1, \dots, q(k)\} : Q_c^k \cap N(v) = \emptyset \}.$$

Grado di saturazione e colorazioni complete

- Il grado di saturazione conta i colori *diversi* fra i vicini colorati di v , e

$$|F_k(v)| = q(k) - sat_k(v).$$

- Un nodo con $U^k = \emptyset$ è una **colorazione completa**: una soluzione ammissibile del VCP con $q(k)$ colori.

I nodi come colorazioni parziali

Un nodo dell'albero è una colorazione parziale; ogni figlio colora un vertice in più.

Non si risolve alcun problema LP: i bound di un nodo vengono dal conteggio dei colori.

Branching

Ora che sappiamo che cos'è un nodo,
decidiamo quale vertice colorare dopo, e con quali colori.

Il vertice di branching

- ▶ Nel nodo k facciamo branching sul vertice non colorato v di **grado di saturazione massimo** $sat_k(v)$
- ▶ I pareggi si risolvono con il $du_k(v)$ massimo, e poi con l'indice minimo
- ▶ È la regola dell'euristica *DSATUR* (Brélaz, 1979)

I figli di un nodo

I figli del nodo k danno un colore a v :

1. un figlio per ogni colore ammissibile $c \in F_k(v)$, in ordine crescente di c ;
 2. un figlio con il **colore nuovo** $q(k) + 1$.
- ▶ I colori $q(k) + 2, q(k) + 3, \dots$ non si usano: un figlio con colore $q(k) + 2$ differirebbe dal figlio con colore $q(k) + 1$ solo per il nome di un colore vuoto
 - ▶ I figli **non sono copie simmetriche** l'uno dell'altro
 - ▶ Il vertice più saturo ha meno figli: $|F_k(v)| + 1$. Se $F_k(v) = \emptyset$, il nodo ha solo il figlio con il colore nuovo

La regola di branching

Il vertice di branching è il più vincolato.

I suoi figli sono i suoi colori ammissibili più un colore nuovo, quindi due figli non descrivono mai la stessa colorazione con i colori rinominati.

Dopo il branching, i bound:
un upper bound da un'euristica, un lower bound da una clique,
e una regola di potatura che conta solo i colori.

Upper bound e clique greedy

- ▶ Il VCP è un problema di minimo: la soluzione incumbent dà l'**upper bound** UB, e un lower bound viene da una clique
- ▶ **Upper bound.** Prima del branching, l'euristica *DSATUR* del capitolo «**Euristiche costruttive**» dà la prima soluzione incumbent, e UB è il suo numero di colori
- ▶ **Clique greedy.** I vertici si scorrono per grado non crescente (pareggi: indice minimo), e un vertice entra in K se è adiacente a tutti i vertici già in K . L'insieme $K = \{k_1, \dots, k_r\}$ è una clique massimale, e

$$LB := r \leq \omega(G) \leq \chi(G).$$

Pre-colorazione e lower bound di un nodo

- ▶ **Pre-colorazione.** I vertici di K richiedono r colori diversi in ogni colorazione, e i nomi dei colori sono arbitrari: il nodo radice colora k_i con il colore i , $i = 1, \dots, r$. Questo rompe la simmetria dei primi r colori
- ▶ **Lower bound di un nodo.** I colori di un nodo non vengono mai tolti nel suo sottoalbero, quindi ogni colorazione completa sotto il nodo k usa almeno $q(k)$ colori:

$$\text{LB}(k) := \max\{q(k), |K|\} = q(k),$$

dove l'uguaglianza vale perché la clique è pre-colorata

Potatura e colorazioni complete

- ▶ **Potatura.** Il numero di colori è un intero, quindi un nodo può portare a una soluzione migliore solo se usa meno di UB colori. Il nodo k è **potato per bound** se

$$q(k) \geq \text{UB}.$$

- ▶ **Colorazione completa.** Un nodo con $U^k = \emptyset$ e $q(k) < \text{UB}$ è una nuova soluzione incumbent: $\text{UB} \leftarrow q(k)$, e i nodi della lista con $q \geq \text{UB}$ vengono rimossi
- ▶ Se $\text{UB} = |K|$, la soluzione incumbent è ottima e l'algoritmo si ferma

I bound dell'algoritmo

DSATUR dà UB, una clique greedy dà LB.

Un nodo è potato appena usa UB colori, perché il suo sottoalbero non può contenere una colorazione con meno colori.

Scelta del nodo

- ▶ L'albero si esplora **in profondità**: la lista L è una pila, i figli di un nodo si valutano in ordine crescente di colore, e il figlio con il colore minimo si estrae per primo
- ▶ I nodi sono numerati nell'ordine di valutazione
- ▶ Un figlio si valuta appena viene creato: la valutazione calcola solo q e controlla se la colorazione è completa
- ▶ Mettiamo ora insieme tutti i pezzi nello pseudocodice dell'algoritmo

L'algoritmo di branch-and-bound DSATUR (1/2)

Input: un'istanza del VCP, cioè un grafo $G = (V, E)$.

Output: una colorazione ottima e il numero cromatico $\chi(G)$.

Passo 0: Inizializzazione

- (a) Eseguire *DSATUR*: porre la soluzione incumbent uguale alla sua colorazione e UB uguale al suo numero di colori.
- (b) Trovare la clique greedy $K = \{k_1, \dots, k_r\}$ e porre $LB \leftarrow r$. Se $LB = UB$, la soluzione incumbent è ottima. STOP.
- (c) Creare il nodo 1 con $col^1(k_i) = i, i = 1, \dots, r$, e $q(1) = r$. Inserire il nodo 1 nella lista L e porre il contatore dei nodi $c \leftarrow 1$.

L'algoritmo di branch-and-bound DSATUR (2/2)

Passo 1: Test di ottimalità

Se L è vuota, UB è il valore ottimo e la soluzione incumbent è ottima. STOP.

Passo 2: Scelta del nodo di branching

Togliere da L il nodo k inserito per ultimo.

Passo 3: Branching

Scegliere $v \in U^k$ con $sat_k(v)$ massimo, poi $du_k(v)$ massimo, poi indice minimo. Per ogni colore $a \in F_k(v) \cup \{q(k) + 1\}$, in ordine crescente, creare il nodo $c + 1$ come il nodo k con $col(v) = a$, valutarlo con la **sotto-procedura del nodo** e porre $c \leftarrow c + 1$. Inserire i figli sopravvissuti in L in ordine decrescente di colore e andare al Passo 1.

La sotto-procedura del nodo

- ▶ Nel Passo 3, il contatore c dei nodi dà il prossimo indice libero, e a è il colore dato a v nel figlio
- ▶ Ogni figlio è valutato dalla **sotto-procedura del nodo**: applica le operazioni di bounding e decide se il nodo sopravvive
- ▶ Il suo pseudocodice è nella prossima slide

La sotto-procedura del nodo: pseudocodice

Input: un nodo h , cioè una colorazione parziale con $q(h)$ colori.

Passo 1: Operazioni di bounding

- (a) Se $q(h) \geq UB$, il nodo è potato per bound. STOP.
- (b) Se $U^h = \emptyset$, aggiornare l'upper bound e la soluzione incumbent: $UB \leftarrow q(h)$. Togliere da L tutti i nodi con $q \geq UB$. Se $UB = LB$, la soluzione incumbent è ottima: STOP dell'algoritmo. Altrimenti STOP.

Passo 2: Sopravvivenza

Il nodo sopravvive: viene inserito in L nel Passo 3. STOP.

L'algoritmo è esatto?

Guardiamo prima l'albero senza potatura:
quali colorazioni contiene, e quante volte?

Proposizione 1: Branching senza simmetrie

Consideriamo l'albero dell'algoritmo di branch-and-bound DSATUR senza potatura. Le sue foglie sono colorazioni complete, e ogni partizione di V in insiemi stabili, cioè ogni soluzione ammissibile del VCP, è l'insieme delle classi di colore di esattamente una foglia.

- La dimostrazione ha due parti: ogni partizione viene raggiunta (**esistenza**), e viene raggiunta una volta sola (**unicità**)

Dimostrazione (1/3)

- ▶ Sia $\mathcal{Q}$ una partizione di V in insiemi stabili
- ▶ I vertici della clique K sono adiacenti a due a due, quindi stanno in r classi diverse; diamo il nome i alla classe di k_i
- ▶ **Esistenza.** Seguiamo un cammino dalla radice mantenendo questo invariante: le classi del nodo sono le classi di $\mathcal{Q}$ ristrette a V^k , ciascuna con il suo nome
- ▶ La radice lo soddisfa. Nel nodo k , siano v il vertice di branching e $Q \in \mathcal{Q}$ la sua classe

Dimostrazione (2/3)

Esistenza (continua). Due casi:

- ▶ Se Q ha già un vertice colorato, con nome c , allora nessun vicino di v ha colore c , perché Q è stabile. Quindi $c \in F_k(v)$, e il figlio con colore c mantiene l'invariante
- ▶ Altrimenti Q non ha ancora vertici colorati: il figlio con il colore nuovo $q(k) + 1$ mantiene l'invariante, e Q prende il nome $q(k) + 1$

Ogni passo colora un vertice, quindi dopo $|U^1|$ passi raggiungiamo una foglia le cui classi sono $\mathcal{Q}$.

Dimostrazione (3/3)

Unicità.

- ▶ Due figli diversi di un nodo mettono v in classi diverse: con i vertici colorati di due colori diversi, oppure con uno di essi e da solo
- ▶ La partizione ristretta a $V^k \cup \{v\}$ è diversa, quindi le foglie sotto due figli diversi danno partizioni diverse



L'algoritmo è esatto

- ▶ **L'algoritmo è esatto.** Un nodo è potato solo se $q(k) \geq UB$, e ogni foglia sotto di esso usa almeno $q(k)$ colori: non si perde alcuna colorazione con meno di UB colori. Quando la lista è vuota, $UB = \chi(G)$
- ▶ Senza la regola del colore nuovo, ogni partizione con q classi comparirebbe una volta per ogni modo di dare nomi alle sue classi con i colori $1, \dots, UB - 1$: l'albero conterrebbe molte copie simmetriche della stessa soluzione

Enumerazione senza simmetrie

Senza potatura, l'albero enumera ogni soluzione ammissibile del VCP esattamente una volta, a meno di una ridenominazione dei colori.

La potatura per bound elimina solo sottoalberi con almeno UB colori: l'algoritmo è esatto.

Riepilogo (1/2)

Nodi:

- ▶ un nodo è una colorazione parziale: vertici colorati V^k , vertici non colorati U^k , $q(k)$ colori usati
- ▶ per $v \in U^k$: grado di saturazione $sat_k(v)$, grado $du_k(v)$, colori ammissibili $F_k(v)$

Branching:

- ▶ sul vertice non colorato con $sat_k(v)$ massimo, poi $du_k(v)$ massimo, poi indice minimo
- ▶ un figlio per ogni colore ammissibile, più un figlio con il colore nuovo $q(k) + 1$

Riepilogo (2/2)

Bound e potatura:

- ▶ UB da *DSATUR*; LB = $|K|$ da una clique greedy, pre-colorata nella radice
- ▶ un nodo è potato per bound quando $q(k) \geq \text{UB}$

Visita e correttezza:

- ▶ visita in profondità: la lista L è una pila
- ▶ senza potatura, ogni colorazione ammissibile è una foglia esattamente una volta: l'algoritmo è esatto

Un esempio svolto

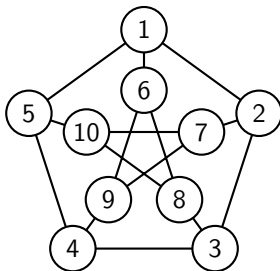
Introduzione: Un esempio svolto

- ▶ In questa sezione eseguiamo a mano l'algoritmo di branch-and-bound DSATUR sul grafo di Petersen
- ▶ Seguiamo le regole della sezione precedente: branching sul vertice più saturo, visita in profondità, figli in ordine crescente di colore
- ▶ L'albero è piccolo, ma dimostra che la colorazione trovata da *DSATUR* è ottima

Esempio: il grafo di Petersen

Consideriamo il grafo di Petersen, l'istanza di riferimento del capitolo «[Problemi di colorazione](#)», con $n = 10$ vertici, $m = 15$ spigoli e tutti i gradi uguali a 3:

$$E = \{ \{1, 2\}, \{1, 5\}, \{1, 6\}, \{2, 3\}, \{2, 7\}, \{3, 4\}, \{3, 8\}, \{4, 5\}, \\ \{4, 9\}, \{5, 10\}, \{6, 8\}, \{6, 9\}, \{7, 9\}, \{7, 10\}, \{8, 10\} \}.$$



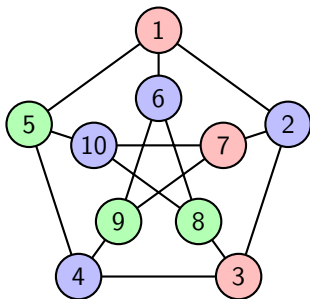
Esempio: inizializzazione (1/3)

L'euristica *DSATUR* (esempio «*DSATUR* sul grafo di Petersen» del capitolo «[Euristiche costruttive](#)»):

- colora i vertici nell'ordine
1, 2, 3, 4, 5, 6, 8, 7, 9, 10
- dà la soluzione incumbent

$$\bar{Q} = \{ \{1, 3, 7\}, \{2, 4, 6, 10\}, \\ \{5, 8, 9\} \},$$

e $UB = 3$



Esempio: inizializzazione (2/3)

- ▶ Tutti i gradi sono uguali, quindi la clique greedy scorre i vertici $1, 2, \dots, 10$
- ▶ Il vertice 1 entra, il vertice 2 è adiacente a 1 ed entra, e nessun altro vertice è adiacente a entrambi

- ▶ Quindi

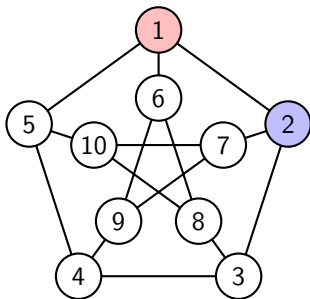
$$K = \{1, 2\} \quad \text{e} \quad \text{LB} = 2$$

- ▶ Il grafo di Petersen non ha triangoli, quindi $\omega(G) = 2$

Esempio: inizializzazione (3/3)

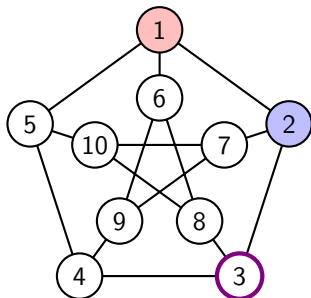
- ▶ Il nodo 1 colora $col^1(1) = 1$ e $col^1(2) = 2$, con $q(1) = 2$
- ▶ Poiché $LB = 2 < 3 = UB$, il nodo 1 viene inserito nella lista L :

$$2 \leq \chi(G) \leq 3.$$



Esempio: iterazione 1

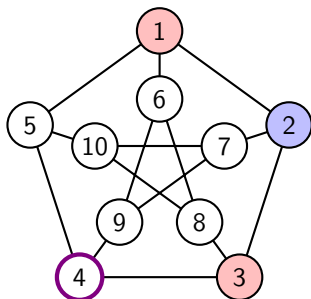
- ▶ Il nodo 1 viene estratto da L
- ▶ I vertici 3, 5, 6, 7 hanno un vicino colorato ciascuno, quindi $sat_1 = 1$; gli altri hanno $sat_1 = 0$
- ▶ I quattro vertici in pareggio hanno $du_1 = 2$, quindi vince l'indice minimo: il vertice di branching è 3
- ▶ Il suo vicino 2 ha colore 2, quindi $F_1(3) = \{1\}$



1. Nodo 2: $col(3) = 1$, con $q(2) = 2 < UB$. Il nodo sopravvive.
2. Nodo 3: $col(3) = 3$, il colore nuovo, con $q(3) = 3 \geq UB$. Il nodo è potato per bound.

Esempio: iterazione 2

- ▶ Il nodo 2 viene estratto da L , con $Q_1 = \{1, 3\}$ e $Q_2 = \{2\}$
- ▶ I vertici 4, 5, 6, 7, 8 hanno $sat_2 = 1$ e $du_2 = 2$, e 9, 10 hanno $sat_2 = 0$: il vertice di branching è 4
- ▶ Il suo vicino 3 ha colore 1, quindi $F_2(4) = \{2\}$



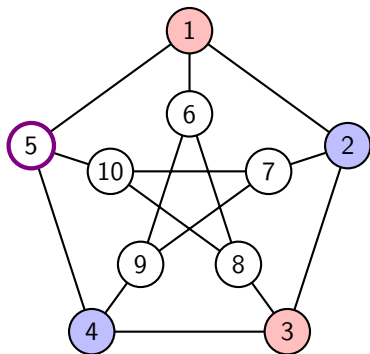
1. Nodo 4: $col(4) = 2$, con $q(4) = 2 < UB$. Il nodo sopravvive.
2. Nodo 5: $col(4) = 3$, con $q(5) = 3 \geq UB$. Il nodo è potato per bound.

Esempio: iterazione 3 (1/2)

- ▶ Il nodo 4 viene estratto da L , con $Q_1 = \{1, 3\}$ e $Q_2 = \{2, 4\}$
 - ▶ Il vertice 5 ha i vicini 1 (colore 1) e 4 (colore 2): $sat_4(5) = 2$, è l'unico vertice con grado di saturazione 2 e quindi è il vertice di branching
 - ▶ I suoi colori ammissibili sono $F_4(5) = \emptyset$
1. Nodo 6: $col(5) = 3$, l'unico figlio, con $q(6) = 3 \geq UB$. Il nodo è potato per bound.

Esempio: iterazione 3 (2/2)

La colorazione parziale del nodo 4 è disegnata qui sotto: il ciclo 1, 2, 3, 4, 5 ha cinque vertici, e due colori non bastano per chiuderlo.



Esempio: terminazione

- ▶ La lista L è vuota: la soluzione incumbent $\bar{Q}$ è ottima e

$$\chi(G) = \text{UB} = 3$$

- ▶ L'albero ha 6 nodi

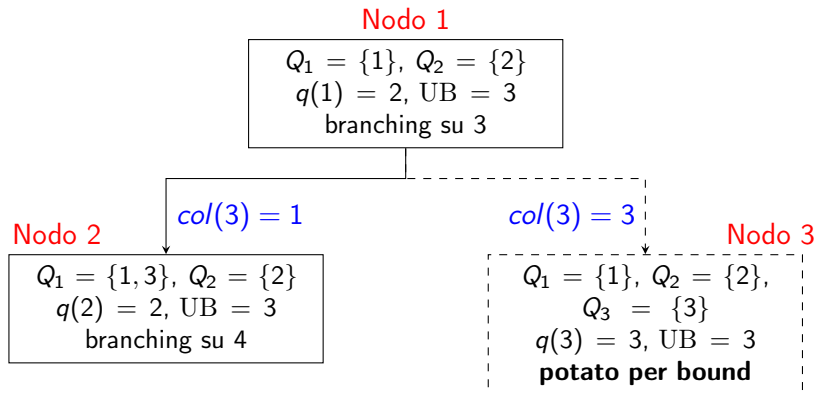
Disegniamo ora l'albero di branch-and-bound, un passo alla volta.

L'albero di branch-and-bound

- ▶ Per ogni nodo, l'albero riporta:
 - ▶ le sue classi di colore
 - ▶ il numero di colori q
 - ▶ l'upper bound UB al momento della valutazione
 - ▶ il vertice di branching o il criterio di potatura
- ▶ I nodi potati sono tratteggiati
- ▶ L'albero è troppo grande per una slide: lo disegniamo un'iterazione alla volta, e poi tutto intero in forma compatta

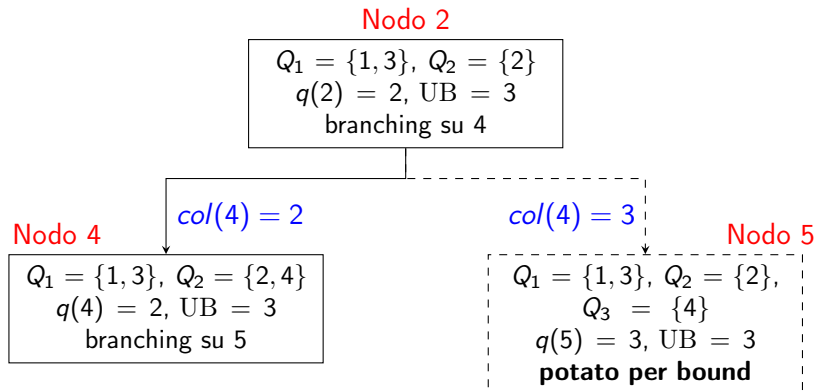
L'albero di branch-and-bound (1/4)

Iterazione 1: il nodo 1 e i suoi figli.



L'albero di branch-and-bound (2/4)

Iterazione 2: il nodo 2 e i suoi figli.



L'albero di branch-and-bound (3/4)

Iterazione 3: il nodo 4 e il suo unico figlio.

Nodo 4

$Q_1 = \{1, 3\}, Q_2 = \{2, 4\}$
 $q(4) = 2, \text{UB} = 3$
branching su 5

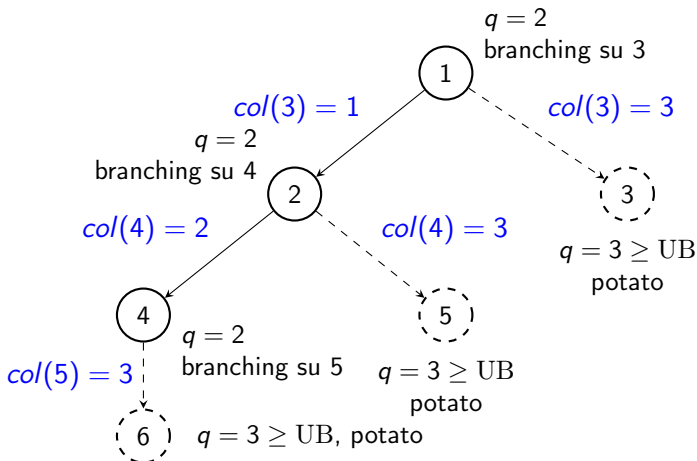
$col(5) = 3$

Nodo 6

$Q_1 = \{1, 3\}, Q_2 = \{2, 4\},$
 $Q_3 = \{5\}$
 $q(6) = 3, \text{UB} = 3$
potato per bound

L'albero di branch-and-bound (4/4)

L'albero intero, in forma compatta: i nodi potati sono tratteggiati.



I nodi nell'ordine di valutazione

Nodo	Padre	Figlio	q	UB	Vertice di branching	Esito
1	–	radice, $K = \{1, 2\}$	2	3	3 ($sat = 1, du = 2$)	branching
2	1	$col(3) = 1$	2	3	4 ($sat = 1, du = 2$)	branching
3	1	$col(3) = 3$	3	3	–	potato per bound
4	2	$col(4) = 2$	2	3	5 ($sat = 2, du = 1$)	branching
5	2	$col(4) = 3$	3	3	–	potato per bound
6	4	$col(5) = 3$	3	3	–	potato per bound

Che cosa mostra l'esempio

- ▶ L'algoritmo non migliora la soluzione incumbent: dimostra che $UB = 3$ è ottimo
- ▶ La dimostrazione è il ciclo dispari 1, 2, 3, 4, 5, trovato dalla ricerca in tre livelli
- ▶ Senza la potatura, l'albero del grafo di Petersen ha 9 776 nodi e 7 430 foglie, una per ogni partizione ammissibile contata nel capitolo «[Problemi di colorazione](#)», come prevede la proposizione del branching senza simmetrie

Inizializzazione:

- ▶ *DSATUR* dà $UB = 3$; la clique greedy $K = \{1, 2\}$ dà $LB = 2$
- ▶ il nodo 1 pre-colora la clique: $2 \leq \chi(G) \leq 3$

Ricerca:

- ▶ tre branching, sui vertici 3, 4 e 5; ogni figlio con $q = 3$ è potato per bound
- ▶ la lista è vuota dopo 6 nodi: $\chi(G) = 3$, dimostrato dal ciclo dispari 1, 2, 3, 4, 5

Osservazioni

Introduzione: Osservazioni

- ▶ In questa sezione vediamo da dove viene l'algoritmo
- ▶ Vediamo anche due raffinamenti della sua regola di branching, che cambiano solo la gestione dei pareggi
- ▶ I riferimenti sono alla fine della sezione

Origine e regola di pareggio di Sewell

- ▶ **Origine.** Il grado di saturazione e l'algoritmo esatto che fa branching su di esso sono dovuti a Brélaz (1979). Pre-colorare una clique è il modo usuale di eliminare la simmetria dei primi colori
- ▶ **Regola di pareggio di Sewell (1996).** Fra i vertici di grado di saturazione massimo, Sewell sceglie il vertice v che massimizza il numero di colori ammissibili in comune con i suoi vicini non colorati,

$$\sum_{u \in N(v) \cap U^k} |F_k(v) \cap F_k(u)|.$$

Colorare quel vertice toglie più opzioni al resto del grafo

- ▶ **PASS (San Segundo, 2012).** PASS applica la regola di Sewell solo a vertici selezionati, calcolando il conteggio sui vertici in pareggio per il grado di saturazione massimo, che costa meno
- ▶ Entrambe le regole cambiano solo la gestione dei pareggi: i figli, i bound e la proposizione del branching senza simmetrie restano gli stessi

- ▶ D. Brélaz, «New methods to color the vertices of a graph», *Communications of the ACM* 22(4), 251–256, 1979
- ▶ E. C. Sewell, «An improved algorithm for exact graph coloring», in D. S. Johnson, M. A. Trick (a cura di), *Cliques, Coloring, and Satisfiability*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science 26, 1996
- ▶ P. San Segundo, «A new DSATUR-based algorithm for exact vertex coloring», *Computers & Operations Research* 39(7), 1724–1733, 2012

Origine:

- ▶ grado di saturazione e algoritmo esatto: Brélaz (1979)
- ▶ pre-colorare una clique elimina la simmetria dei primi colori

Gestione dei pareggi:

- ▶ Sewell (1996): il vertice con più colori ammissibili in comune con i suoi vicini non colorati
- ▶ PASS (San Segundo, 2012): il conteggio di Sewell solo sui vertici in pareggio
- ▶ figli, bound e proposizione del branching senza simmetrie non cambiano

Errori tipici

Introduzione: Errori tipici

- ▶ Raccogliamo gli errori che si fanno più spesso con l'algoritmo di branch-and-bound DSATUR
- ▶ Due riguardano il grado di saturazione e i figli, due la potatura, due la clique
- ▶ Controllateli prima di eseguire a mano l'algoritmo

Errori tipici (1/3)

Osservazione 1: Contare i vicini colorati invece dei colori

Il grado di saturazione conta i colori diversi attorno a v : due vicini con lo stesso colore contano una volta sola.

Osservazione 2: Un figlio per ogni colore $1, \dots, UB - 1$

I colori non usati sono tutti equivalenti: solo il primo colore nuovo $q(k) + 1$ dà un figlio, altrimenti l'albero contiene copie simmetriche della stessa colorazione.

Errori tipici (2/3)

Osservazione 3: Dimenticare il colore nuovo

Il figlio con colore $q(k) + 1$ si crea sempre, anche quando $F_k(v) \neq \emptyset$; senza di esso alcune colorazioni non si raggiungono mai.

Osservazione 4: Potare con $q(k) > \text{UB}$

Un nodo con $q(k) = \text{UB}$ non può portare a una colorazione migliore, perché il numero di colori è un intero: la regola è $q(k) \geq \text{UB}$.

Osservazione 5: Pre-colorare un insieme che non è una clique

Solo i vertici di una clique richiedono colori diversi a due a due in ogni colorazione; pre-colorare con colori diversi due vertici non adiacenti può escludere tutte le colorazioni ottime.

Osservazione 6: Continuare dopo $UB = |K|$

Quando una colorazione completa raggiunge il bound della clique, è ottima: i nodi ancora nella lista non si esplorano.

Grado di saturazione e figli:

- ▶ il grado di saturazione conta i colori diversi, non i vicini colorati
- ▶ i figli sono i colori ammissibili più il colore nuovo $q(k) + 1$, sempre e solo quello

Potatura e clique:

- ▶ potare quando $q(k) \geq UB$, non solo quando $q(k) > UB$
- ▶ pre-colorare solo una clique, e fermarsi appena $UB = |K|$

In sintesi

Introduzione: In sintesi

- ▶ Raccogliamo le idee chiave del capitolo sull'algoritmo di branch-and-bound DSATUR
- ▶ Il quadro riassuntivo raccoglie i bound su $\chi(G)$ trovati finora per il grafo di Petersen



Riepilogo (1/2)

- ▶ Un nodo del branch-and-bound DSATUR è una colorazione parziale; l'algoritmo fa branching sul vertice più saturo, con un figlio per ogni colore ammissibile e uno per un colore nuovo.
- ▶ *DSATUR* dà il primo upper bound, una clique greedy dà il lower bound ed è pre-colorata; un nodo è potato quando $q(k) \geq \text{UB}$.
- ▶ Senza potatura l'albero enumera ogni colorazione ammissibile una volta, a meno di una ridenominazione dei colori: l'algoritmo è esatto.

Riepilogo (2/2)

Quadro riassuntivo del grafo di Petersen ($n = 10$, $m = 15$). I bound su $\chi(G)$ trovati finora:

Fonte	Lower bound	Upper bound	Da
Bound della clique («Problemi di colorazione»)	2	–	$\omega(G) = 2$
<i>DSATUR</i> («Euristiche costruttive»)	–	3	$\{1, 3, 7\}$, $\{2, 4, 6, 10\}$, $\{5, 8, 9\}$
Questo capitolo, inizializzazione	2	3	clique greedy $K = \{1, 2\}$
Questo capitolo, terminazione	3	3	lista vuota, 6 nodi