Ragionamento Automatico: Tree Decomposition

Brian Riccardi

Contents

Chapter 1

Il problema

Il problema che vogliamo risolvere è la ricerca di una *tree decomposition* di *width* minima di un grafo non orientato G. Verranno utilizzate le tecniche di programmazione con vincoli (Minizinc) e di programmazione di tipo Answer Set (ASP/clingo).

1.1 Definizioni e osservazioni

Procediamo dando una definizione formale del problema e una serie di definizioni che utilizzeremo in seguito.

Definizione. 1. *(**Tree Decomposition**)*
Sia $G = (V_G, E_G)$ un grafo non orientato, $T = (V_T, E_T)$ un albero e $\chi : V_T \longrightarrow \mathscr{P}(V_G)$. Diremo che $\tau = (T, \chi)$ è una tree decomposition per G se e solo se:

TD1: $\bigcup_{t \in V_T} \chi(t) = V_G$

TD2: *Per ogni arco $uv \in E_G$ esiste $t \in V_T$ tale che $\{u, v\} \subseteq \chi(t)$*

TD3: *Per ogni terna $r, s, t \in V_T$ tale che s si trovi nel cammino da r a t, vale $\chi(r) \cap \chi(t) \subseteq \chi(s)$*

Definizione. 2. *(**Width**)*
*Data una tree decomposition $\tau = (T, \chi)$, definiamo **treewidth** la quantità $w(\tau) = \max_{t \in V_T}\{|\chi(t)|\} - 1$.*

Definizione. 3. *(**Minimal tree decomposition**)*
Dato un grafo G non orientato, il problema della minimal tree decomposition è la ricerca di una tree decomposition per G di width minima. Se tale decomposition ha width $w(\tau) = k$, allora diremo che G ha treewidth $w(G) = k$.

Da questo punto in poi daremo per scontato che ogniqualvolta si parli di una tree decomposition, questa sia riferita a un grafo G non orientato di n nodi e che la decomposition sia $\tau = (T, \chi)$.

Il concetto di tree decomposition può essere visto come una variante del problema della colorazione: possiamo infatti immaginare i nodi dell'albero come a dei colori e la funzione χ come a un'assegnamento dei colori ai nodi. In particolare, per (**TD3**) viene richiesto che ogni nodo del grafo venga colorato con un sottoalbero di T e per (**TD2**) viene richiesto che i rispettivi sottoalberi si intersechino. Dato questo cambio di prospettiva spesso parleremo di *colori* quando ci riferiremo ai nodi di T.

La prima osservazione che uno può fare è che non esiste una decomposition unica; inoltre, non è immediato dare un limite superiore a $|V_T|$ tale che T induca una decomposition minimale. In particolare, tale bound è utile ai nostri scopi: ci permette di definire dei modelli (a vincoli o answer set) più efficaci.

Diamo ancora qualche definizione che ci sarà utile nella prossima sezione.

Definizione. 4. *Dato un albero $T = (V, E)$ e un arco $uv \in E$, definiamo la* ***contrazione*** *di uv come l'albero T/uv dove i nodi u, v vengono rimpiazzati da un nodo $\overline{uv}$ e gli archi incidenti a u e v diversi da uv diventano ora incidenti a $\overline{uv}$. Data una tree decomposition τ e un arco $uv \in E_T$, la contrazione di uv induce $\chi_{uv} : V_{T/uv} \longrightarrow \mathscr{P}(V_G)$ definita da*

$$\chi_{uv}(t) = \begin{cases} \chi(t) & \text{se } t \neq \overline{uv} \\ \chi(u) \cup \chi(v) & \text{se } t = \overline{uv} \end{cases}$$

Diremo che uv è τ-contraibile se e solo se la coppia $(T/uv, \chi_{uv})$ è una decomposition[1] *di treewidth* ***non maggiore****.*
Diremo inoltre che τ è contraibile se ammette un arco τ-contraibile, ***completamente contratto*** *altrimenti.*

1.2 Lemmi e teoremi

In questa sezione ci preoccuperemo di derivare la bound menzionata precedentemente; vedremo infatti che un qualsiasi grafo di n nodi ammette una tree decomposition minimale di al più n colori.

Lemma. 1.
Siano r, s colori tali che $\chi(r) \subseteq \chi(s)$. Allora rs è τ-contraibile.

Proof. Banale. □

[1] Non è difficile verificare che le decomposition sono chiuse per contrazione.

Lemma. 2. *Sia τ una decomposition di $m > n$ colori, allora esistono r, s tali che $\chi(r) \subseteq \chi(s)$.*

Proof. Supponiamo per assurdo che tali colori non esistano e fissiamo una foglia l di T e il suo vicino h. Per ipotesi, esiste $x \in \chi(l)$ tale che $x \notin \chi(h)$. Consideriamo dunque una visita in profondità dell'albero a partire da l con lo scopo di enumerare i nodi di G. Tale enumerazione sarà la successione $\{\sigma_i\}_{i \in 1 \dots m}$

In particolare:

- $\sigma_1 := x$
- Al passo $k > 1$ della visita, quando entro in un nuovo colore s arrivando da r, $\sigma_k := y$ tale che
$$\begin{cases} y \in \chi(s), \\ y \notin \chi(r), \\ \forall i < k (y \neq \sigma_i) \end{cases}$$

L'esistenza di un tale y è assicurata dall'ipotesi iniziale e da (**TD3**): se non esistesse un tale y, si avrebbe $\chi(r) \supseteq \chi(s)$ oppure si avrebbe $\sigma_j = y$ per qualche $j < k - 1$, ma allora verrebbe meno (**TD3**) in quanto y avrebbe due colori distinti ma non ogni colore "intermedio" (ovvero r, per costruzione).

Abbiamo quindi costruito una sequenza $\{\sigma_1, \sigma_2, \dots, \sigma_m\}$ di nodi distinti di G, ma questo è un assurdo in quanto per il Principio della Piccionaia — essendo $m > n$ — almeno un nodo compare due volte in tale successione.

□

I due lemmi, ci assicurano che qualunque sia una decomposition, se questa ha un numero sufficiente di colori allora sarà contraibile.

Teorema. 1. ***(Bounded decomposition)***
Sia G un grafo di treewidth $w(G) = k$, allora esiste una tree decomposition per G di al più n colori.

Proof. Sia τ una tree decomposition minimale per G con $m > n$ colori. Dai lemmi precedenti sappiamo che τ è contraibile e dunque, applicando $m - n$ contrazioni sugli archi che soddisfano l'ipotesi del lemma 1, otteniamo una nuova tree decomposition minimale τ' di n colori. □

1.3 Idea risolutiva

Il teorema della sezione precedente ci assicura che se ci concentriamo su decompositions di al più n colori non rischiamo di perderci la soluzione ottima. I modelli delle sezioni successive, quindi, si baseranno su questa idea.

Analizzando la definizione di decomposition balza subito all'occhio (**TD3**) che richiede una trattazione non banale: data una terna di colori, saper decidere

se un colore appartiene al path tra gli altri due.

Sebbene i due modelli utilizzino paradigmi differenti, l'idea alla base per la decisione di (**TD3**) si basa sul concetto di *Lowest Common Ancestor*[2]; infatti, supponendo di aver radicato l'albero: siano r, s, t i colori, e $\mathcal{A}$ la relazione *antenato di*, allora

$$s \in \pi_{rt} \iff s = LCA(r,t) \vee \big(s\mathcal{A}r \oplus s\mathcal{A}t\big)$$

Viene quindi naturale radicare l'albero e, per eliminare qualche simmetria:

- Imporre sempre al nodo 1 il ruolo di radice
- Enumare i colori dall'alto verso il basso imponendo $\text{parent}(x) < x$ e $(x < y \longrightarrow depth(x) \leq depth(y))$

Chapter 2

Minizinc

Minizinc è un linguaggio per **modellare** problemi esprimendoli come un insieme di domini, variabili e vincoli su queste variabili.

2.1 Struttura del modello

Nel modello sono presenti le costanti `N` e `graph` che rappresentano, rispettivamente, n e la matrice di adiacenza di G. Si è poi deciso di inserire una variabile intera `n` di dominio $[1, \texttt{N}]$ per rappresentare $|V_T|$. Sono presenti un insieme di variabili `parent`, `depth` e `lca` di ovvia interpretazione, `ancestor` che decide se un colore è antenato di un altro (utile per vincolare `lca`). Infine, sono presenti delle variabili binarie `hasColor` per decidere se un nodo ha un determinato colore.

[2]Ricordiamo che, in un albero radicato, il Lowest Common Ancestor tra due nodi x, y è (l'unico) nodo $z = LCA(x, y)$ tale che x, y sono discendenti di z e z è massimale (rispetto alla profondità) tra i nodi che godono di questa proprietà.

La prima serie di constraints ha lo scopo di vincolare `parent` e `depth` per ottenere a tutti gli effetti un albero. Si preoccupano di applicare le idee risolutive discusse precedentemente. In particolare, il predicato `non_decreasing_except_0` è stato definito da me.

La seconda serie di constraints si occupa di vincolare la relazione di `ancestor` (che verrà utilizzata per il calcolo di `lca`). I constraints non sono complessi da capire: ci preoccupiamo di vincolare a `0` tutto ciò che riguarda colori *extra* (ovvero più grandi di `n`) e ci preoccupiamo di vincolare gli antenati *ovvi* (parent e figlio) ed esprimiamo una condizione necessaria e sufficiente (simil-induttiva) per vincolare un antenato al suo discendente. Viene poi impiegato un vincolo ridondante per assicurarci che un colore con indice grande non possa essere considerato antenato di un colore con indice piccolo.

La terza serie di vincoli si preoccupa di costruire la tabella `lca`: dopo la solita *pulizia* per rendere noto al solver che siamo interessati solo agli LCA di colori $\leq$`n`, sono state inserite delle condizioni per assicurarci che `lca[x, y] <= min{x, y}`, delle condizioni per vincolare la simmetria `lca[x, y]=lca[y, x]` e delle condizioni simil-induttive per il calcolo effettivo, nonché un vincolo per il calcolo di LCA quando un colore è antenato dell'altro.

Vengono poi espresse le tre condizioni di tree decomposition: ogni nodo ha un colore, due nodi adiacenti condividono un colore e se un nodo ha due colori distinti, allora deve avere tutti i colori presenti nel percorso tra essi; per quest'ultimo, è stato definito un predicato `inPath`. Sono stati poi definiti dei vincoli per assicurarci di usare ogni colore dell'albero e di non usare alcun colore al di fuori dell'albero. Viene espressa la variabile `objective` e vincolata ad essere la cardinalità massima (meno 1) degli insiemi indotti dalla colorazione. Infine, vengono vincolati i colori affinché ogni $\chi(t)$ abbia cardinalità almeno la metà dell width nella speranza di aiutare il solver a considerare meno casi (talvolta inutili): la correttezza del vincolo è data dal lemma sulle "inclusioni".

2.2 Search strategies

Sono state provate diverse search strategies ma nessuna ha sortito grandi effetti sulla qualità della soluzione. Le strategie proposte in questa relazione riguardano la scelta dei valori di `parent` e `depth`: in una viene applicata `int_search(parent ++ depth, first_fail, indomain_min, complete)`, mentre nell'altra `int_search(parent ++ depth, first_fail, indomain_random, complete)`. Nel primo caso, si cerca di prediligere alberi il più possibile "com-

patti", nel secondo caso invece si cerca di non assumere alcuna euristica.

Chapter 3

ASP/clingo

In un programma Answer Set vengono descritti un insieme di fatti e un insieme di regole di deduzione al fine di produrre un *insieme stabile*.

3.1 Struttura del modello

Una relazione binaria `edge` descrive gli archi del grafo sottoforma di fatti (si è scelto di utilizzare solo uno dei due versi), con una costante `n` che rappresenta n. Sono presenti relazioni binarie `parent` e `hasDepth`, unarie `root` e `nt` per descrivere l'albero: in particolare, `nt` determina il numero di nodi dell'albero. Vi sono relazioni binarie `hasColor` e `ancestor`, ternarie `lca` e `inPath` di ovvi significati.

Una serie di default permette di dedurre che ogni colore ha un'unica profondità (positiva nel caso in cui il colore non sia `root`) e questa profondità è strettamente legata alla relazione `parent`; anche in questo caso si è deciso di eliminare qualche simmetria introducendo dei default che forzassero il modello ad assumere albero indicizzati dall'alto verso il basso.

Una seconda serie di default permette di descrivere le condizioni per una tree decomposition e per descrivere le relazioni `lca` e `inPath`: entrambe vengono descritte facilmente, esprimendo simmetrie e definizioni induttive; in particolare, `inPath` sfrutta la solita proprietà descritta precedentemente. Infine, viene definita una relazione binaria `size` per descrivere le cardinalità dei χ, e una relazione unaria per descrivere la funzione obbiettivo.

3.2 Search strategies

Sono stati effettuati due tipi di test: in uno è stato utilizzato `clingo` *standard*, nell'altro è stata utilizzata l'opzione `--opt-strategy=bb` che cerca di stabilire se una soluzione trovata è ottima "dimostrando" l'insoddisfacibilità di modelli

di costo minore: questo approccio mi è sembrato interessante avendo a che fare con un problema dalle condizioni stringenti e dipendenti dalla struttura stessa del grafo (intuitivamente, più il grafo è denso e più la width è alta).

Chapter 4

Risultati sperimentali

I test sono stati effettuati su macchina Windows con processore i5-8250U@1.60GHz, 8 GB di memoria.

I modelli e le rispettive strategie sono stati testati su una batteria di 100 grafi non orientati e connessi generati casualmente; in particolare, i grafi sono stati generati a scaglioni:

- 20 grafi di $n = 10$ nodi
- 20 grafi di $n = 20$ nodi
- 20 grafi di $n = 30$ nodi
- 10 grafi di $n = 40$ nodi
- 10 grafi di $n = 50$ nodi

Per ogni scaglione i grafi hanno acquisito, in ordine crescente, un numero m di archi nell'intervallo $\left[n-1, \frac{n(n-1)}{2}\right]$. Sono stati tenuti in considerazione il valore della decomposition trovata dopo 5 minuti di esecuzione e, per quanto riguarda i modelli ASP/clingo, si è considerato anche il numero di "risposte" per ottenere tale risultato. Gli input sono stati numerati da 0 a 99 in senso crescente rispetto all'ordinamento indotto da (n, m).

Le tabelle riportano i risultati trovati da ASP senza e con strategia (`Opt basic` e `Opt BB` con i rispettivi numeri di risposta) e i risultati trovati da Minizinc nelle due strategie (`S1` e `S2`).

Per quanto riguarda il confronto "interno" si può notare che, generalmente, la strategia `bb` batte la `basic` sia in *qualità* della soluzione, sia nel numero di "tentativi" per trovarla; per Minizinc, l'euristica sugli alberi compatti definita da `S1` viene spesso battuta dalla strategia random di `S2`.

Infine, da un confronto "esterno" si evince che il modello ASP è notevolmente più efficiente, trovando soluzioni migliori anche per grafi più grandi; si può notare infatti che per grafi sopra i 20 nodi Minizinc non è più capace di trovare soluzioni.

Input	Opt basic	#ans basic	Opt BB	#ans BB	S1	S2
0	1	7	1	9	1	1
1	2	4	2	7	2	2
2	1	7	1	9	2	1
3	2	6	2	8	2	2
4	2	7	2	8	4	2
5	3	5	3	7	4	3
6	3	6	3	7	4	3
7	4	6	4	6	4	4
8	4	6	4	5	6	4
9	4	6	4	6	5	4
10	5	4	5	5	5	5
11	5	5	5	5	7	5
12	6	4	6	4	6	6
13	5	5	5	5	6	5
14	6	4	6	4	7	6
15	6	4	6	4	7	6
16	7	3	7	3	7	7
17	8	2	8	2	8	8
18	8	2	8	2	8	8
19	9	1	9	1	9	9
20	3	17	3	17	12	4
21	2	18	2	18	--	4
22	4	15	4	16	--	7
23	5	13	5	15	8	5
24	6	14	6	14	12	8
25	8	12	8	12	12	11
26	9	11	9	11	12	12
27	9	11	9	11	14	--
28	11	9	11	9	--	14
29	11	9	11	9	15	--
30	12	8	11	9	16	--
31	12	8	12	8	16	--
32	13	7	13	7	17	17
33	14	6	14	6	17	15
34	14	6	14	6	16	16
35	16	4	16	4	17	--
36	16	4	16	4	17	17
37	16	4	16	4	17	--
38	17	3	17	3	18	18
39	19	1	19	1	19	--

Input	Opt basic	#ans basic	Opt BB	#ans BB	S1	S2
40	3	23	3	20	--	--
41	5	24	4	26	--	--
42	10	20	9	21	--	--
43	14	15	12	18	--	--
44	16	13	14	16	--	--
45	15	14	16	14	--	--
46	17	12	17	13	--	--
47	18	12	18	12	--	--
48	20	10	20	10	--	--
49	21	9	21	9	--	--
50	21	9	22	8	--	--
51	21	9	21	9	--	--
52	23	7	23	7	--	--
53	24	6	23	7	--	--
54	24	5	24	6	--	--
55	24	6	24	6	--	--
56	26	4	25	5	--	--
57	25	5	26	4	--	--
58	27	3	27	3	--	--
59	29	1	29	1	--	--
60	3	37	3	37	--	--
61	13	27	12	28	--	--
62	17	23	16	24	--	--
63	21	19	20	20	--	--
64	23	13	24	16	--	--
65	26	14	26	14	--	--
66	28	12	28	12	--	--
67	30	10	30	9	--	--
68	30	10	31	9	--	--
69	30	10	31	9	--	--
70	32	6	31	9	--	--
71	33	7	32	8	--	--
72	33	6	33	7	--	--
73	34	6	34	6	--	--
74	35	5	34	6	--	--
75	35	5	35	5	--	--
76	36	4	35	5	--	--
77	36	4	36	4	--	--
78	37	3	37	3	--	--
79	39	1	39	1	--	--

Input	Opt basic	#ans basic	Opt BB	#ans BB	S1	S2
80	20	30	18	32	--	--
81	34	15	31	19	--	--
82	36	13	35	14	--	--
83	39	10	40	8	--	--
84	40	9	41	9	--	--
85	43	7	43	7	--	--
86	44	6	44	6	--	--
87	46	3	45	5	--	--
88	47	3	46	4	--	--
89	49	1	49	1	--	--
90	36	22	30	24	--	--
91	45	14	41	17	--	--
92	51	9	46	14	--	--
93	55	4	51	8	--	--
94	55	4	52	8	--	--
95	55	5	51	8	--	--
96	55	5	54	6	--	--
97	56	4	56	4	--	--
98	58	2	56	4	--	--
99	59	1	59	1	--	--