7. VISUALIZZAZIONE DI CARTE GEOGRAFICHE

7.1. Premessa

7.1.1. Problemi NP-completi e riduzioni polinomiali

Riferimenti: [CLR] pp 864-877, 883-885 esclusa dim. Th. Di Cook.

7.1.2. Un algoritmo per 2-SAT

Riferimenti: [P94] pp 183-187.

Definizione. Una formula proposizionale è in *forma normale congiuntiva* se è della forma $A_1 \wedge \ldots \wedge A_m$, dove per $1 \le i \le m$, A_i (detta anche *clausola*) è della forma $B_{i,1} \vee \ldots \vee B_{i,h_i}$, dove per $1 \le j \le h_i$, $B_{i,j}$ è un *letterale*.

Definizione. Una formula proposizionale si dice *soddisfacibile* se assume valore di verità true per almeno un assegnamento delle sue variabili.

Il problema del *k-SAT* (dove SAT sta per "soddisfacibilità", dall'inglese satisfiability) consiste nel trovare un assegnamento delle variabili di una data formula proposizionale in forma normale congiuntiva le cui clausole siano formate da *k* letterali ($k \ge 1$) che la soddisfi.

Per $k \ge 3$ la versione decisionale del problema è NP-completo, ma per $k = 2$ il problema è risolvibile in tempo polinomiale, come dimostrato nel seguito.

Data una formula proposizionale ϕ in forma normale congiuntiva in cui tutte le clausole siano formate da due letterali, dobbiamo trovare un'assegnamento *T* delle sue variabili che soddisfi ϕ. A tal scopo, a partire da un'istanza ϕ di 2SAT, definiamo un grafo $G(\phi)$ come segue:

- i nodi di *G* sono le variabili di ϕ e le loro negate;
- *G* contiene un arco orientato (α, β) se e solo se esiste in ϕ una clausola $(\neg\alpha \vee \beta)$ oppure $(\beta \vee \neg\alpha)$ (notare che la clausola $\neg\alpha \vee \beta$ genera due archi: (α, β), $(\neg\beta, \neg\alpha)$ data l'equivalenza con $(\neg\alpha \vee \neg(\neg\beta))$. Di conseguenza, *G* possiede una simmetria: se (α, β) è un arco di G, allora lo è anche $(\neg\beta, \neg\alpha)$.

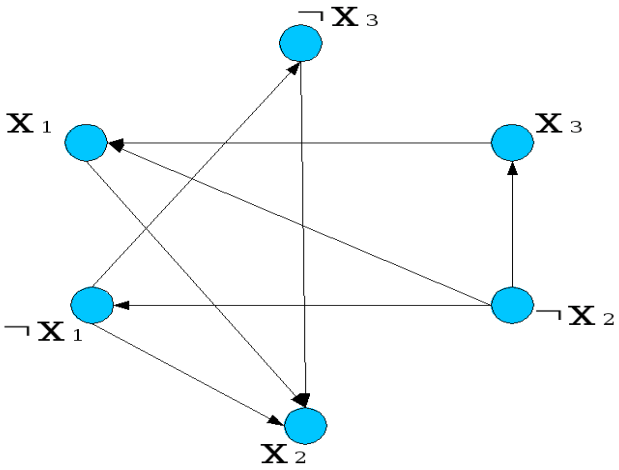
Esempio. Sia ϕ la formula seguente: $(x_1 \vee x_2) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_2) \wedge (x_2 \vee x_3)$.
Il grafo generato è mostrato in Figura 7.1.

Intuitivamente, gli archi rappresentano le implicazioni logiche di ϕ (infatti l'implicazione $\alpha \Rightarrow \beta$ può essere riscritta come $\neg\alpha \vee \beta$). Perciò, i cammini in $G(\phi)$ sono delle implicazioni logiche (per la transitività di $\Rightarrow$).

Dimostriamo ora il seguente:

Teorema: *ϕ non è soddisfacibile se e solo se esiste una variabile x tale che ci siano i cammini da x a $\neg x$ e da $\neg x$ a x in $G(\phi)$.*

Figura 7.1.



Dimostrazione: Dimostriamo l'implicazione $\Leftarrow$, cioè: esistono i cammini $\Rightarrow \phi$ non è soddisfacibile. Supponiamo per assurdo che ϕ possa essere soddisfatta da un qualche assegnamento di verità T. Indicato con $T(x)$ il valore del letterale x nell'assegnamento di verità T, supponiamo che $T(x)$=true (una dimostrazione del tutto simile può essere prodotta per $T(x)$=false). Poiché esiste un percorso da x a $\neg x$ per ipotesi, e dato che da $T(x)$=true segue che $T(\neg x)$=false, allora deve esistere un arco (α, β) lungo questo cammino tale che $T(\alpha)$=true e $T(\beta)$=false. Da questo deriva che, dato che (α, β) è un arco di $G(\phi)$, necessariamente $(\neg\alpha \vee \beta)$ è una clausola di ϕ. A questo punto però cadiamo in un assurdo dato che la clausola non è soddisfatta da T (infatti sostituendo si ha che $\neg T(\alpha) \vee T(\beta) =$ false).

Dimostriamo ora l'implicazione $\Rightarrow$, cioè: ϕ non soddisfacibile $\Rightarrow$ esistono i cammini; invece di questo asserto dimostriamo, equivalentemente, il suo negato: non esistono i cammini $\Rightarrow \phi$ soddisfacibile. Supponiamo allora che non esista alcuna variabile x tale che ci siano i cammini da x a $\neg x$ e da $\neg x$ a x in $G(\phi)$ e costruiamo un assegnamento di verità che soddisfi ϕ, cioè tale che nessun arco di $G(\phi)$ passi da true a false. Prendiamo un nodo α, il cui valore non sia ancora stato definito, e tale che non ci siano percorsi da α a $\neg\alpha$ (per ipotesi) e assegniamo a tale nodo il valore true. Consideriamo ora tutti i nodi in $G(\phi)$ raggiungibili da α e assegniamo loro valore true**.** Assegnamo valore false alla negazione di questi nodi (cioè a tutti i nodi dai quali $\neg\alpha$ è raggiungibile). Ripetiamo questo passaggio fino a che tutti i nodi abbiano un assegnamento di verità.

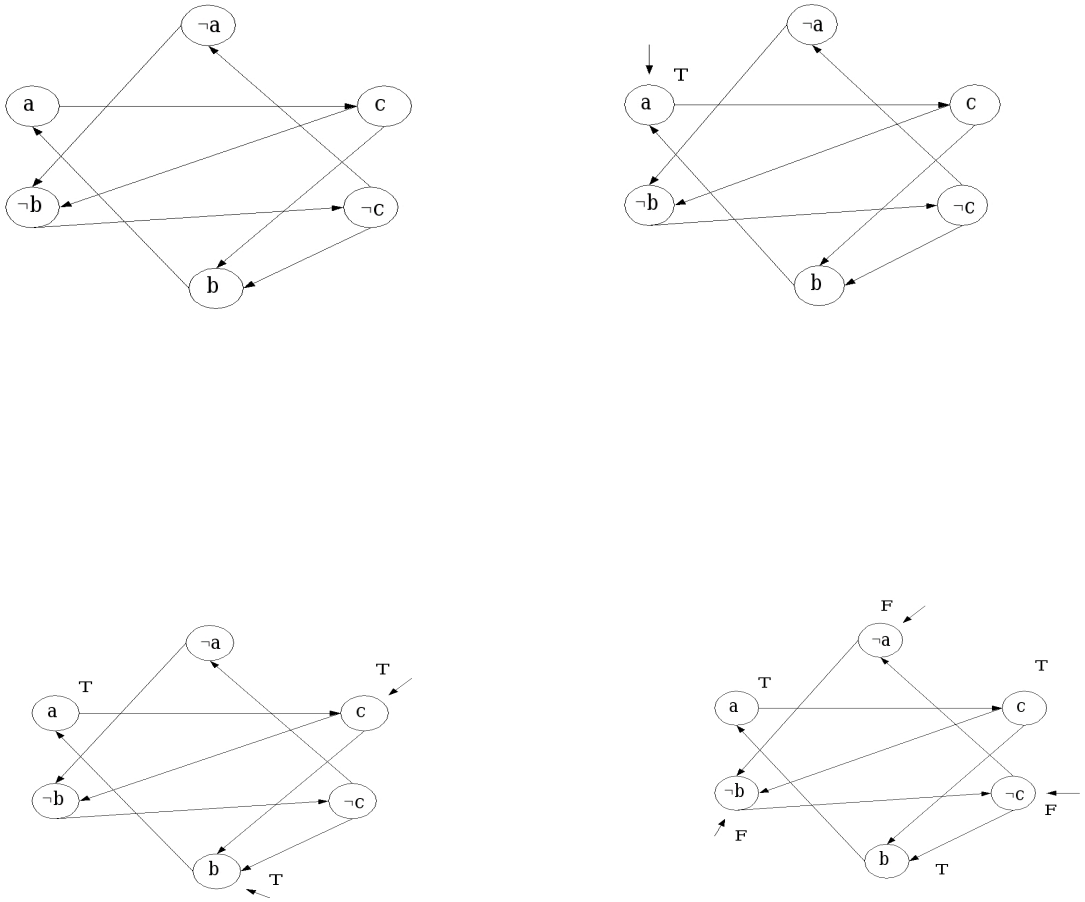
Questo passo è ben formulato, cioè non possono esistere i percorsi dal nodo α ad un nodo β e dal nodo α al nodo $\neg\beta$, poiché se esistessero, allora esisterebbero anche i cammini da β e da $\neg\beta$ a $\neg\alpha$ (per la simmetria di $G(\phi)$), e di conseguenza esisterebbe un percorso da α a $\neg\alpha$ (passando per β), il che contraddice le ipotesi.

Poiché abbiamo assunto che non esistono percorsi da nessun x a $\neg x$ e viceversa, a tutti i nodi sarà possibile dare un valore di verità, inoltre per come è stato definito il passaggio, ogni volta che ad un nodo viene assegnato true, anche ai suoi predecessori viene assegnato lo stesso valore, e discorso simmetrico può essere fatto se l'assegnamento è false, quindi non esistono archi che vanno da vero a falso. Allora l'assegnamento di verità soddisferà ϕ. **CVD**

Esempio.

In Figura 7.2 è riportata una esecuzione dell'algoritmo spiegato nella dimostrazione per la formula $(\neg a \vee b) \wedge (\neg b \vee c) \wedge (a \vee \neg c) \wedge (c \vee b)$

Figura 7.2.



E' immediato convincersi del fatto che la dimostrazione ora fornita è costruttiva, cioè consente, data una formula, di risolvere il problema di 2-SAT, se possibile, altrimenti dire che esso non ammette soluzione. Se utilizziamo banalmente l'algoritmo proposto (per ogni x verificare che non esista il cammino verso $\neg x$, la complessità è $O(n(n+m))$, tuttavia si può dimostrare che la soluzione al problema del 2-SAT si può dare in tempo $O(n)$.

7.2. Il problema della etichettatura degli oggetti

Riferimenti: [KW01] pp 247-273 esclusi par. 10.3.5, 10.3.6 e 10.6

attenzione: da qui fino alla fine del capitolo sono solo enumerati i concetti in modo molto conciso. Per studiare, fare riferimento al testo

Problema di visualizzare dati (sotto forma di etichette) insieme agli oggetti. Questo problema, detto *automatic map labeling* si generalizza nella problematica del *graph labeling*.

Tipicamente, l'input è un disegno e un insieme di etichette, l'output è un disegno etichettato.

3 tipi di oggetti da etichettare:

- punti: città, vette, luoghi particolari, vertici di grafi o di diagrammi;

- linee: fiumi, confini, strade, archi rettilinei o curvi di grafi o digrafi;
- aree: isole, regioni, laghi, facce di grafi.

L'etichettatura di aree è facile (basta mettere l'etichetta dentro l'area) e quindi non la tratteremo, mentre ci occuperemo dell'etichettatura di punti e linee.

Alcuni cartografi, come Imholf e Yoeli (anni '70) hanno tentato di dare regole per misurare la chiarezza semantica di una etichettatura di una carta geografica. Estraiamo 3 concetti comunemente accettati come regole di base per etichettare bene non solo carte geografiche ma anche grafi:

- leggibilità: la misura del carattere deve essere sufficientemente grande;
- non ambiguità: ogni etichetta deve essere univocamente associata all'oggetto cui si riferisce;
- non sovrapposizione: ne' tra etichette ne' con oggetti della mappa o del grafo.

Le possibili posizioni sono dette *etichette candidate*, e ad esse si può associare un *costo*, che ne riflette la qualità rispetto alle regole appena elencate.

7.3. Panoramica della complessità dei principali problemi di etichettatura

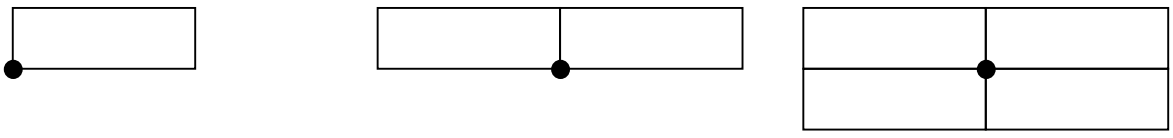
La maggior parte dei problemi definiti nell'area dell'*automatic map labeling* sono NP-completi, e quindi i ricercatori tentano di fornire approssimazioni ed euristiche. Vediamo nel seguito i principali problemi e le loro complessità.

7.3.1. Etichettatura di punti

Il modo di etichettare dipende dal modello di etichettatura. I più importanti sono:

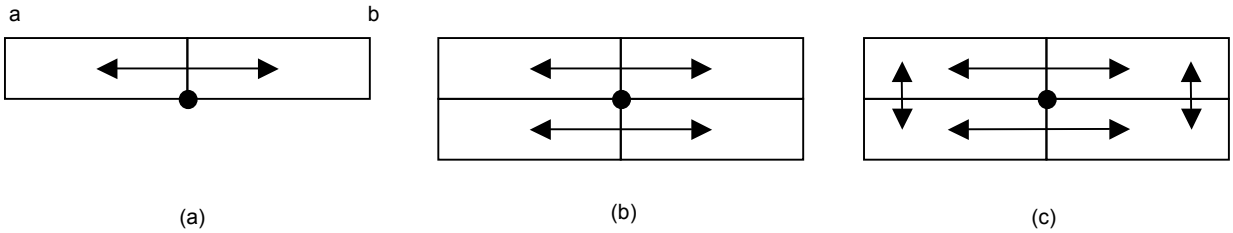
- modello a posizione fissa: ogni oggetto ha un insieme finito di etichette candidate. Possiamo avere il modello a posizione fissa ad 1, 2, o 4 etichette candidate (vedi fig. 7.3).

Figura 7.3.



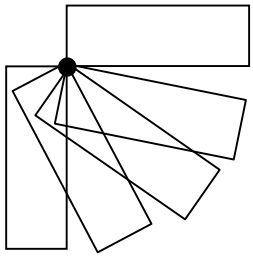
- modello a posizione fissa con etichette scalabili: come prima, ma la dimensione delle etichette può essere scalata
- modello ad etichette scorrevoli: l'etichetta può essere messa in ogni posizione che tocchi l'oggetto (le etichette possono essere *shiftate* in modo continuo – vedi fig. 7.4).

Figura 7.4.



- Alternativamente, le etichette possono essere ruotate (vedi fig. 7.5).

Figura 7.5.



Problemi tipici:

PROBLEMA DI DECISIONE (DP)

Esiste un assegnamento di etichette di dimensione fissata *s* tale che ogni oggetto sia etichettato con un'etichetta scelta nel suo insieme delle etichette candidate senza alcuna sovrapposizione?

PROBLEMA DI ETICHETTATURA (LP)

Se la risposta al problema di decisione è si, trova un assegnamento di etichette di dimensione fissata *s* tale che ogni etichetta sia scelta nel suo insieme delle etichette candidate senza alcuna sovrapposizione

MASSIMIZZAZIONE DEL NUMERO DI ETICHETTE (MAX#P)

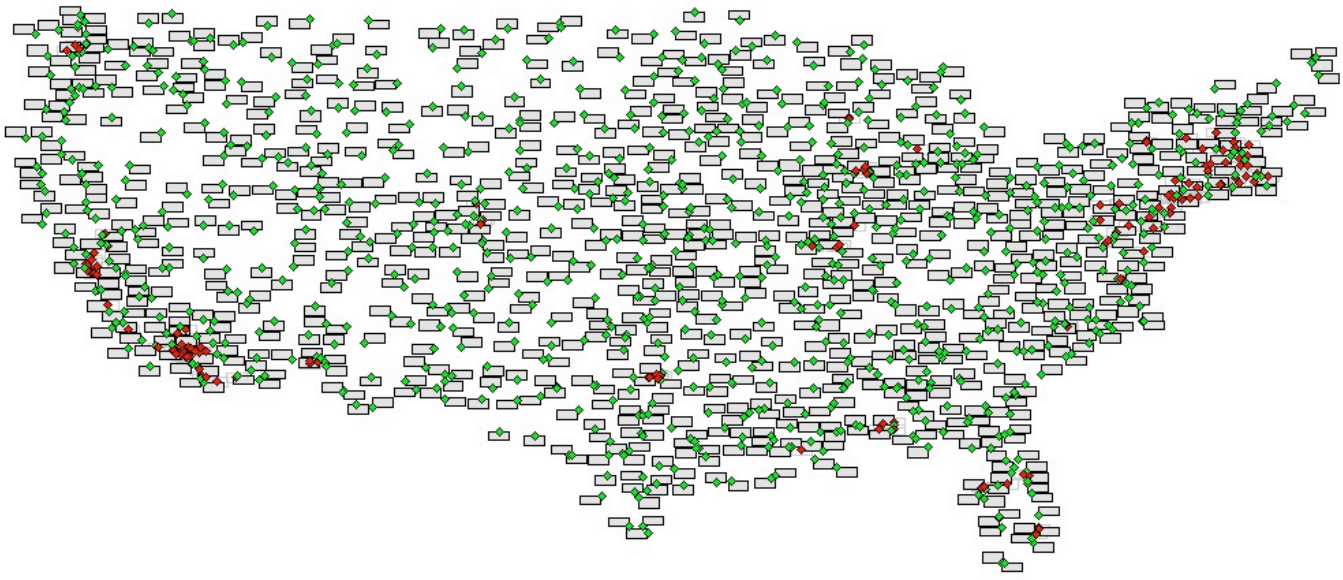
Assegna quante più etichette possibile di dimensione fissata *s* tale che ogni etichetta sia scelta nel suo insieme delle etichette candidate senza alcuna sovrapposizione. Si veda, ad esempio, la Figura 7.6, incui è stato utilizzato il modello ad etichette scorrevoli, ed i punti rossi rappresentano le città che in nessun modo possono essere etichettate senza creare sovrapposizioni.

MASSIMIZZAZIONE DELLA DIMENSIONE DELLE ETICHETTE (MAXsizeP)

Trova un fattore di scalatura *s* massimo e un corrispondente assegnamento di etichette tale che ogni etichetta sia scelta nel suo insieme delle etichette candidate senza alcuna sovrapposizione.

N.B. la differenza tra *map labeling* e *graph labeling*: in quest'ultimo posso decidere di muovere leggermente gli oggetti per fare entrare meglio le etichette, nelle carte geografiche questo non si può fare!

Figura 7.6.



Vediamo ora le relazioni tra i veri problemi:

- LP ≥ DP: ovvio
- LP ≥ MAXsizeP: supponiamo di avere un algoritmo *A* che risolva il problema LP, allora con una ricerca binaria posso trovare la massima dimensione.
- MAXsizeP ≥ LP e MAXsizeP ≥ DP: analogamente al caso precedente, un algoritmo che risolve MAXsizeP risolve anche LP semplice e ovviamente DP.
- MAX#P ≥ LP: ovvio

Da queste relazioni si deduce, in particolare, che MAXsizeP ed LP sono nella stessa classe di complessità (informalmente, sono ugualmente difficili), visto che LP ≥ MAXsizeP ≥ LP.

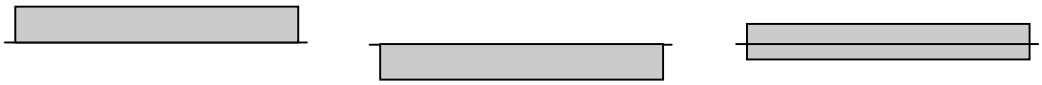
7.3.2. Etichettatura di linee

Nel caso di carte geografiche, etichettare le linee risulta molto difficile perché il più delle volte non sono linee rette. Hanno senso però anche problemi più semplici, ad esempio che considerano linee rette orizzontali o verticali, perché funzionano bene per disegni di grafi ortogonali.

MASSIMIZZAZIONE DEL NUMERO DI ETICHETTE CON LINEE ORIZZONTALI (LHLP)

Il problema consiste nel considerare un insieme di segmenti orizzontali (o verticali), ed etichettarli con un rettangolo di altezza fissata e lungo quanto tutto il segmento, se questo è possibile. Per ciascuna etichetta sono possibili 3 posizioni (vedi fig. 7.7).

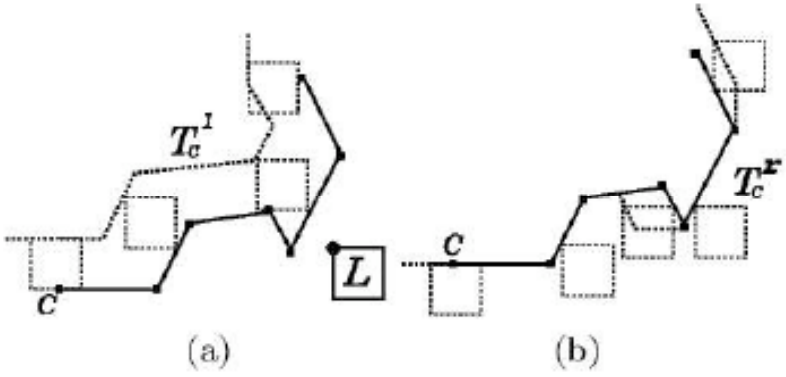
Figura 7.7.



Nel caso più generale, invece, una curva è rappresentata da una serie di segmenti connessi fra loro, chiamiamo questo insieme di segmenti *catena*.

Per ciò che riguarda l'etichetta di un punto abbiamo preso in considerazione il rettangolo che racchiude l'etichetta, nel caso invece di etichettatura di curve possiamo alternativamente considerare di nuovo i rettangoli o, per aumentare la precisione, i quadrati che racchiudono ogni singola lettera dell'etichetta e l'etichetta è un insieme di quadrati. In quest'ultimo caso, definiamo *spazio di etichettatura* lo spazio in cui il quadrato si può muovere lungo la catena senza sovrapporsi ad altri punti. Definiamo *punto di riferimento* del quadrato il suo vertice in alto a sinistra. In figura 7.8 la linea tratteggiata rappresenta l'insieme di tutte le possibili posizioni che il punto di riferimento può occupare sopra o sotto i segmenti muovendo il quadrato lungo la catena.

Figura 7.8.

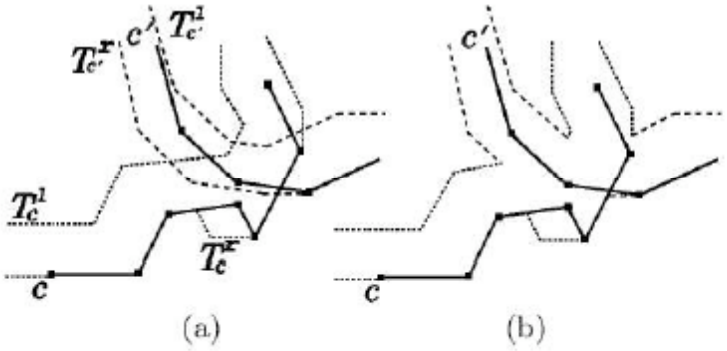


A questo punto assegniamo una direzione alla catena, per esempio da sinistra in basso a destra in alto, e chiamiamo T_c^l (T_c^r) la spezzata di tutte le possibili posizioni del punto di riferimento a sinistra (destra) della catena.

Se due catene si sovrappongono e si intersecano anche i loro spazi di etichettatura si sovrappongono (figura 7.9.a), e le intersezioni fra i due spazi di etichettatura vengono cancellati (figura 7.9.b). Può accadere che, pur se due catene non si incrociano, i loro spazi di etichettatura si intersechino; anche in questo caso rimuoviamo le intersezioni.

Il problema di etichettatura consiste nel posizionare tutte le lettere all'interno dello spazio di etichettatura, ma in questo corso non ci occuperemo di questo problema. Accenneremo, invece, al problema di etichettare linee curve nel caso in cui l'etichetta sia un singolo rettangolo.

Figura 7.9.



7.4. Alcuni algoritmi di etichettatura

7.4.1. Etichettatura di punti

Descriviamo, ora, alcuni algoritmi di etichettatura.

LP

Formann e Wagner [FW91] hanno studiato il problema dell'etichettatura di punti nel modello a 4 posizioni, così definito:

Definizione. Dato un insieme di punti nel piano, lo scopo del *problema di etichettatura di punti a 4 posizioni* è quello di etichettare ciascun punto con un rettangolo a lati paralleli agli assi cartesiani tale che un angolo coincida con il punto da etichettare, e non ci siano sovrapposizioni tra rettangoli.

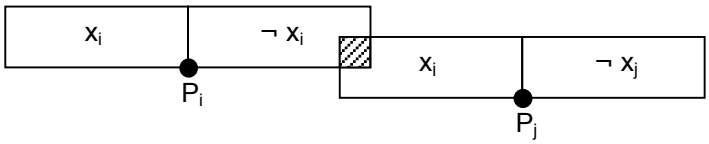
La versione decisionale di questo problema è NP-completa, anche se le etichette sono quadrate e tutte della stessa misura. Questo risultato è stato ottenuto indipendentemente da Kato e Imai [KI88] e da Marks e Shieber [MS91]. La dimostrazione di NP-completezza si basa sulla riduzione da 3-SAT, provando che il problema di etichettatura ha una soluzione senza sovrapposizioni se e solo se la formula è soddisfacibile.

Formann e Wagner hanno pure mostrato che il modello a 2 posizioni si risolve in tempo polinomiale ($O(n)$), riducendolo a 2-SAT nel modo seguente:

- i punti diventano le variabili della formula; se l'etichetta è a sx del punto i allora ho la variabile x_i, altrimenti ho la sua negata.

- Siano P_i e P_j due punti (figura 7.10), se è possibile una sovrapposizione tra l'etichetta destra di P_i ($\neg x_i$) e l'etichetta sinistra di P_j (x_j) allora questa può essere evitata se $\neg x_i$ e x_j non sono entrambe vere. Quindi si costruisce la clausola $\neg((\neg x_i) \wedge x_j) \equiv (x_i \vee \neg(x_j))$. Si può costruire una tale clausola per ogni coppia di etichette candidate incompatibili.

Figura 7.10.



$$\neg((\neg x_i) \wedge x_j) \equiv (x_i \vee \neg(x_j))$$

E' piuttosto intuitivo che ad un assegnamento di verità per la formula corrisponde un assegnamento fattibile di etichette e, viceversa, se un assegnamento di verità non esiste, allora i punti non possono essere etichettati completamente.

MAXsizeP

Da questo risultato si è cercato di generalizzare al modello a 4 posizioni in cui le etichette vanno massimizzate in dimensione (MAXsizeP). Tale problema è NP-arduo, e si può dimostrare che non esiste un'approssimazione migliore di 2, a meno che P=NP. Wagner [W94] ha dimostrato che esistono algoritmi di approssimazione con rapporto di approssimazione 2 che può essere eseguito in tempo Omega(n log n), nel caso particolare di etichette quadrate.

Idea dell'algoritmo: costruisco le 4 etichette candidate con dim. Minima per ogni punto; ingrandisco via via la dimensione e cancello una delle due etichette che eventualmente si possono intersecare (scelgo a caso quale delle due). Proseguo in questo modo fino a quando ogni punto non ha solo 2 etichette candidate. A questo punto applico la riduzione precedente.

Questo risultato è interessante dal punto di vista teorico, perché dimostra l'approssimabilità del problema, ma è inapplicabile in pratica, perché il rapporto di approssimazione 2 per la dimensione delle etichette è moltissimo!!! Inoltre, questo algoritmo funziona solo per etichette quadrate.

L'algoritmo ora descritto non tiene conto dell'esistenza di archi. Questi possono solo peggiorare la situazione: in presenza di archi, si dimostra che il rapporto di approssimazione rimane 2, ma il tempo computazionale peggiora.

MAX#P

Agarwal, vanKreveld e Suri [AKS98] hanno formulato questo problema in termini di *massimo insieme indipendente* (dato $G=(V,E)$, un *insieme indipendente* I è un sottoinsieme di V, $I \subseteq V$, tale che comunque si scelgano 2 nodi in I, questi non devono essere adiacenti. Un insieme indipendente banale è quello costituito da un nodo singolo; lo scopo del problema è quello di massimizzare la cardinalità di I):

per ogni punto vorrei metter un'etichetta. Costruisco un grafo in cui ho tanti nodi quante sono le possibili etichette candidate; metto un arco tra 2 nodi se le etichette corrispondenti hanno intersezione non vuota (N.B. le etichette candidate di uno stesso punto formano una clicca). In questo modo, stiamo riducendo il problema al problema del massimo insieme indipendente, che è NP-arduo. Si può dimostrare che l'algoritmo ora descritto per MAX#P è *log n*-approssimante. Nel caso in cui le etichette abbiano altezza unitaria, esiste una PTAS, che è (1+1/k)-approssimante e può essere eseguito in tempo $O(n \log n + n^{2k-1})$.

MAX#P con etichette scorrevoli

Iturriaga e Lubiw [IL97] hanno dimostrato che il problema è NP-arduo già con un solo grado di libertà (cioè quando l'etichetta può scorrere, ad esempio, solo orizzontalmente). Van Kreveld, Strijk e Wolff [KSW98] hanno confrontato i 3 modelli scorrevoli (ad 1, 2 e 4 gradi di libertà) con quelli a posizione fissa, tentando di rispondere alla domanda: "quanti più punti si possono etichettare usando un modello anziché un altro?" Per questa quantificazione, hanno definito il *rapporto tra due modelli*, come segue:

Definizione. Sia *P* un insieme di punti del piano. Siano *M1* ed *M2* due modelli per etichettare *P*, e siano *opt(M1,P)* e *opt(M2,P)* il massimo numero di punti di *P* che ricevono un'etichetta nei due modelli. Il *rapporto tra i modelli M1 ed M2* è definito come il max su tutti i P di fissata cardinalità n del sup per n che tende all'infinito del rapporto tra opt(M1,P) e opt(M2,P).

Gli autori hanno mostrato limitazioni inferiori e superiori per i rapporti di molte coppie di modelli (Figura 7.11.: vedi fig. 10.4 p. 257 [KW98]). Ad esempio, il massimo numero di punti etichettati con il modello a 2 posizioni è al più il doppio di quello ad una posizione. Questi valori permettono di progettare algoritmi approssimanti per quasi tutti i modelli.

Vediamo ora un algoritmo 2-approssimante per il modello a 4 posizioni (complessità $O(n \log n)$).

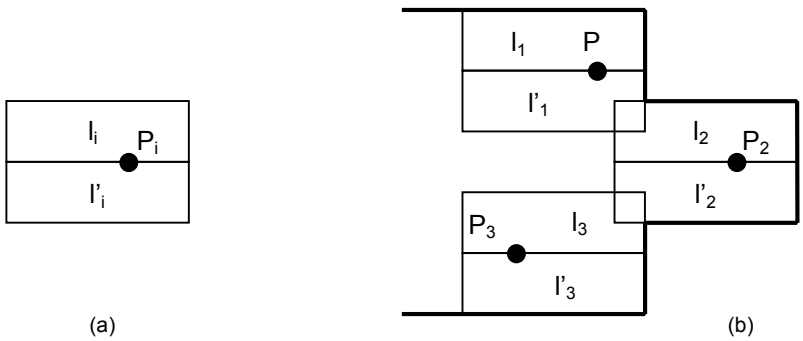
Questo algoritmo lavora sotto l'ipotesi che tutte le etichette abbiano altezza uguale. Dato un insieme *P* di *n* punti, all'*i*-esimo passo si supponga di avere già etichettato *i-1* punti (quelli più a sinistra) e si etichetti l'*i*-esimo punto. L'algoritmo segue la filosofia greedy, e sceglie l'*etichetta più a sinistra,* cioè l'etichetta candidata più a sinistra tra tutte quelle possibili dei punti non etichettati. In questo modo l'algoritmo etichetta almeno la metà dei punti che si possono etichettare e, per farlo, si serve della seguente definizione:

Definizione. L'*involucro destro* delle etichette di un insieme di punti P è una funzione f (figura 7.12.b) definita come $f(y)=max\{-\infty, max\{x:(x,y) \text{ in } l_i \text{ o } l'_i\}\}$ dove:

l_i è l'etichetta già assegnata al punto P_i;

l'_i è la copia di l_i posta più in basso (figura 7.12.a).

Figura 7.12.



La ragione per cui si utilizza questa definizione è che, in questo modo, dell'intera etichetta, posso considerare solo il suo spigolo in basso a sinistra, osservando che la posso aggiungere all'insieme di quelle già scelte se tale spigolo cade alla sinistra dell'involucro destro. E' possibile mantenere bassa la complessità dell'algoritmo mantenendo delle opportune strutture dati (3 heap).

N.B. Questo algoritmo funziona con rapporto di approssimazione 2 solo per rettangoli di altezza costante e lunghezza variabile; esso si può generalizzare ad etichette più generiche, ma non si può più garantire il rapporto di approssimazione.

MAXsizeP con etichette ruotate

Doddi, Marathe, Mirzaian, Moret e Zhu [DMal99] hanno studiato il problema con etichette quadrate e tutte uguali, collegate al punto in una posizione qualsiasi del contorno e con qualsiasi direzione, tentando di massimizzare la dimensione. Essi danno un algoritmo con approssimazione $\frac{8\sqrt{2}}{\sin(\pi/10)}$ in tempo O(n log n). Questo algoritmo ha puro interesse teorico, dimostrando che il problema è approssimabile entro una costante, ma non può essere in alcun modo approssimato, essendo tale costante circa 2000.

7.4.2. Etichettatura di linee

Mostreremo ora che il problema LHLP si risolve polinomialmente, riducendolo a 2-SAT nel modo seguente. Si consideri l'illustrazione di figura 7.11:

per ogni linea *i* si introducono 2 variabili x_{i1} e x_{i2} dove:

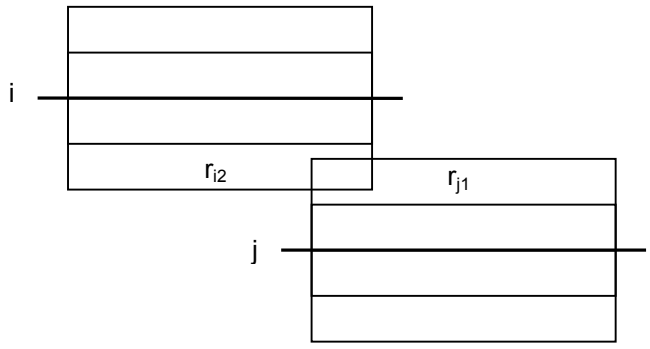
- $x_{i1}=1 \Leftrightarrow r_{i1}$ è usato $\Leftrightarrow r'_{i1}$ non è usato;
  $x_{i1}=0$ altrimenti;
- $x_{i2}=1 \Leftrightarrow r_{i2}$ è usato $\Leftrightarrow r'_{i2}$ non è usato;
  $x_{i2}=0$ altrimenti.

Figura 7.11.

Se si utilizza r_{i1} significa che si sta utilizzando anche r_{i2}, quindi vale $x_{i1} \Rightarrow x_{i2}$ che si può scrivere $x_{i1} \Rightarrow x_{i2} \equiv (\neg (x_{i1}) \vee x_i)$.

Inoltre, siano dati ad esempio x_{j1} e x_{i2} come mostra la figura 7.12, allora le variabili vengono composte nel seguente modo: $\neg (x_{j1} \wedge \neg (x_{i2})) \equiv (\neg (x_{j1}) \vee x_{i2})$.

Figura 7.12.



Usando clausole come questa possiamo codificare tutti i vincoli. Una soluzione di 2-SAT implica, ovviamente, una soluzione al problema e viceversa.

Si osservi che, se togliamo il vincolo che i segmenti siano paralleli agli assi possiamo adattare l'algoritmo ma la complessità peggiora.

Mettiamoci ora nel caso più generale in cui le linee siano curve. L'algoritmo presentato, dovuto a Edmonson e Christensen [EC96], divide il problema in tre sottoproblemi:

1. *Scegliere le etichette candidate*. Data una linea curva, identificare le possibili posizioni per un'etichetta.
2. *Valutare le etichette*. Data una posizione calcolare velocemente il punteggio associato ad una posizione.
3. *Scegliere le etichette*. Scegliere le posizioni delle etichette tale che massimizzino la qualità dell'etichettatura.

Il primo passo si occupa di scegliere le posizioni candidate; si possono identificare tre caratteristiche che deve avere un buon algoritmo che genera posizioni candidate: i. deve scegliere un numero limitato di candidati; ii. deve trovare un buon compromesso tra qualità delle posizioni scelte ed efficienza, infatti scegliere solo posizioni di alta qualità risulterebbe di difficile e lenta risoluzione; iii. le posizioni candidate per un'etichetta non devono essere tutte concentrate in una sola zona ma devono essere disposte nel modo più uniforme possibile intorno all'oggetto da etichettare. Tra le infinite etichette ammissibili per una linea ne scegliamo una certa quantità, in modo che le etichette candidate siano più o meno equidistanti tra di loro, e tale distanza è prefissata; è chiaro che la quantità di etichette scelte in questo modo dipende dalla lunghezza della linea e dell'etichetta stessa. Se tale quantità è comunque troppo alta, si calcola una funzione di costo semplificata (che tiene conto solo delle sovrapposizioni tra etichette candidate), e si eliminano le etichette candidate in eccesso tra quelle con costo più alto.

Il secondo passo consiste nell'assegnare un costo a ciascuna etichetta candidata; a questo costo contribuiscono vari punti:

a. sovrapposizioni; un'etichetta può sovrapporsi con punti, con linee o con altre etichette; il costo dovrà essere tanto maggiore quanto più punti l'etichetta interseca, quanto più è lungo il segmento della linea sulla quale si sovrappone, e quante più sono le etichette con le quali si sovrappone;
b. canoni di valutazioni per il posizionamento di etichette di punti; nel caso di etichette di punti si è osservato che: i. è preferibile che l'etichetta sia a destra; ii. è preferibile che sia sopra piuttosto che sotto; iii. è preferibile che si trovi sulla stessa linea dei punti che si etichettano.
   Rispetto a questi canoni sono state valutate 19 posizioni possibili per l'etichettatura di un punto, e a ciascuna è stata assegnata una valutazione che contribuisce al suo costo;
c. canoni di valutazioni per il posizionamento di etichette di linee; senza entrare in troppi dettagli, sottolineiamo che, nel caso delle etichette di linee, è cruciale la distanza media dell'etichetta dalla linea; inoltre, poiché la linea non è rettilinea, potrebbe avere delle parti tangenti all'etichetta, o che addirittura le si sovrappongono, per questo viene introdotto un parametro che tiene conto della distanza minima; inoltre, una particolare preferenza viene assegnata alle posizioni in cui la linea da etichettare è piuttosto piana; infine, è da preferirsi un'etichetta sopra piuttosto che sotto la linea, ed il più centrata possibile.

Il terzo passo viene risolto tramite la tecnica del *simulated annealing*, di cui abbiamo già discusso.

7.5. Un algoritmo di etichettatura di carte geografiche

L'etichettatura delle carte geografiche prevede l'etichettatura di tutti gli oggetti presenti. In questo contesto facciamo la semplificazione che le etichette siano di dimensione variabile, ma rettangolari con i lati paralleli agli assi. Kokoulis e Tollis [KT98] hanno presentato un algoritmo

approssimante. Viste le condizioni richieste, questo algoritmo funziona particolarmente bene per grafi ortogonali.

Prima di descrivere l'algoritmo, definiamo il *costo* di un'etichetta candidata, valore non negativo che riflette il suo "punteggio" in termini di non ambiguità e di numero di sovrapposizioni con altri oggetti ed etichette. La funzione obiettivo da ottimizzare è la somma dei costi delle etichette scelte.

L'algoritmo si compone di 3 passi:

1. per ogni oggetto, si generano tutte le etichette candidate e si assegna loro un costo;
2. si costruisce un grafo associato $G_R=(V_l,E_l)$ dove ogni nodo $l \in V_l$ rappresenta l'etichetta candidata; dati l_1, $l_2 \in V_l$, se le etichette corrispondenti si sovrappongono allora si costruisce l'arco $(l_1, l_2) \in E_l$. L'obiettivo è ora quello di trovare un sottografo costituto da sole cricche. In ciascuna cricca, solo un'etichetta può essere scelta. Questo ha senso perché tutte le etichette candidate relative allo stesso punto formano una cricca. L'ideale sarebbe trovare le cricche di dimensione massima, però la determinazione della massima cricca è un problema NP-arduo. Quindi, euristicamente, si cancellano nodi in modo tale che rimangano solo cricche. Sia G'_R il grafo ottenuto da tali cancellazioni.
3. si costruisce un grafo bipartito, $G_m=(V_f \cup V_c, E_m)$ dove ogni oggetto (nodo o linea) è un nodo di V_f, e ogni cricca è un nodo di V_c; si inserisce un arco (f,c) solo se esiste un'etichetta candidata di f che appartiene a una cricca in V_c. Ora, dobbiamo trovare un accoppiamento massimale di G_m perché ogni nodo in V_c sia adiacente ad una sola cricca, altrimenti 2 oggetti diversi avrebbero etichette che si intersecano; quindi, tra tutti gli accoppiamenti massimi scegliamo quello di costo minimo; questo può essere fatto in tempo $O(n^{2.5})$.

Si osservi che se il grafo G'_R è quello di partenza, questo accoppiamento trova la soluzione ottima, ma se G'_R si ottiene da G_R rimuovendo casualmente dei nodi, l'algoritmo è approssimante.

7.6. Il problema dell'intersezione di segmenti

Riferimenti: [Bal97] pp 19-29

Osservare una carta geografica può essere difficile, se vi sono rappresentate troppe informazioni. Per questo i sistemi di informazione geografica (GIS) la separano in ciò che vengono chiamati *layers*. Ogni layer è una carta tematica e memorizza solo un tipo di informazione. Avremo, dunque, un layer che memorizza le strade, uno che memorizza le città, uno laghi e fiumi, e così via. I temi di ogni layers possono anche essere astratti, ad esempio un layer potrebbe memorizzare la densità della popolazione, le precipitazioni medie, i tipi di vegetazione.

L'utente di un GIS può selezionare un layer per studiarlo, ma può anche fondere le informazioni contenute in due o più layers per avere un quadro d'insieme. Ad esempio, mettendo insieme la carta stradale e quella in cui sono rappresentate le città, si può avere un'idea di quale percorso fare per raggiungere la propria meta. In tal caso, è di particolare rilievo il problema di intercettare le intersezioni tra gli oggetti visualizzati nei due layers.

Nel seguito studieremo la forma più semplice del problema, in cui le due carte da sovrapporre sono di fatto delle reti (si pensi ad esempio alla carta delle strade, dei fiumi, delle ferrovie, e così via) rappresentate come collezioni di segmenti (dato che le curve possono essere approssimate con delle spezzate). Lo scopo è quello di trovare le intersezioni tra le due reti. Una domanda che ci si può porre è se i segmenti debbano essere aperti o chiusi, se cioè un'intersezione agli estremi sia da considerarsi tale oppure no. I segmenti vanno considerati chiusi infatti se, ad esempio, intersechiamo la carta stradale con quella dei fiumi, è chiaro che una strada interseca un fiume (ad esempio grazie ad un ponte) se la parte interna di essi interseca ma, poiché l'estremo di un segmento potrebbe corrispondere ad un punto interno della linea, essendo questa stata approssimata con una sequenza di segmenti, ne segue che i segmenti vanno considerati chiusi, e bisogna tener presente anche le intersezioni tra estremi.

Per risolvere il problema della sovrapposizione di reti, possiamo formalizzarlo in un contesto geometrico, come segue:

Dati due insiemi di segmenti chiusi, calcolare tutte le intersezioni tra un segmento di un insieme ed un segmento dell'altro insieme.

Per semplificare il problema, è possibile mettere insieme tutti i segmenti in un unico gruppo e cercare le intersezioni tra una qualsiasi coppia di segmenti. A posteriori, se si determina una intersezione tra due segmenti dello stesso gruppo (cosa che accadrà senz'altro perché segmenti consecutivi intersecano ai loro estremi), tale punto dovrà essere eliminato dalla soluzione.

In questa formulazione il problema non sembra così complicato, poiché è sufficiente scorrere tutte le possibili coppie di segmenti e calcolare se ciascuna coppia porta ad un'intersezione oppure no. Se i segmenti sono n, la complessità di questo approccio è $\Theta(n^2)$. Nel caso peggiore, in cui cioè ogni coppia di segmenti ha un'intersezione, questo algoritmo è ottimo. Tuttavia, in pratica, le intersezioni sono un certo numero k, di solito molto più piccolo di n^2; in tal caso la complessità è troppo elevata. Vogliamo, quindi, ottenere un algoritmo la cui complessità dipenda non solo dal numero n di segmenti, ma anche dal numero k di intersezioni trovate. Un tale algoritmo prende il nome di *output-sensitive*, visto che la sua complessità dipende da quello che sarà l'output.

Per progettare un tale algoritmo, possiamo sfruttare la situazione geometrica: se due segmenti sono l'uno in prossimità dell'altro, essi sono dei candidati per una possibile intersezione, ma se essi sono posizionati molto lontano tra loro, allora in nessun modo essi potranno intersecare. Dato un insieme di segmenti $S=\{s_1, s_2, ..., s_n\}$, per evitare di testare l'intersezione di segmenti lontani tra loro, diamo le seguenti definizioni:

Definizione: Dato un segmento s non parallelo all'asse delle ascisse, il suo *estremo superiore (inferiore)* è l'estremo con ordinata maggiore (minore). L'*intervallo-y* di s è la sua proiezione sull'asse y. Se due segmenti hanno i loro intervalli-y disgiunti, allora essi sono detti *lontani*, e non potranno intersecarsi.

Ne segue che, per testare l'intersezione tra due segmenti, basta considerare quelli che hanno i rispettivi intervalli-y che si intersecano, cioè per cui esiste una linea orizzontale che li intercetti entrambi. Per individuare tali coppie introduciamo nel piano una linea orizzontale l che spazza il piano dall'alto verso il basso. Lo *stato di l all'istante t* è l'insieme dei segmenti che essa interseca all'istante t, perciò lo stato varia man mano che, col passare del tempo, l si sposta verso il basso, ma lo fa in modo discreto, variando solo in certi punti, detti *punti evento*. Ovviamente i punti evento sono gli estremi dei segmenti di S: quando viene intercettato un estremo inferiore, significa che l sta abbandonando il segmento corrispondente e questo deve essere eliminato dallo stato di l; se, invece, viene intercettato un estremo superiore, allora l sta intersecando un nuovo segmento, la cui intersezione deve essere testata con tutti i segmenti nello stato di l. Nel seguito, ometteremo l'istante t, essendo esso generico.

Sfortunatamente questo approccio non è sufficiente a garantire che l'algoritmo sia output sensitive: ci sono infatti dei casi in cui si testa comunque l'intersezione di $O(n^2)$ coppie, anche se le intersezioni trovate saranno molte meno, ad esempio se S è costituto da n segmenti verticali tutti posizionati alla stessa altezza: l li intercetta contemporaneamente tutti e quindi verifica le intersezioni tra tutte le coppie, ma non ne trova. La ragione di questo problema è che viene verificata l'intersezione tra segmenti che, di fatto, sono situati lontani l'uno dall'altro.

Per ovviare a ciò, ordiniamo i segmenti da sinistra a destra appena essi entrano nello stato di l. In questo modo vogliamo inglobare l'idea di vicinanza anche rispetto all'asse x, oltre che y. Con questa variante, i punti evento non sono più solo gli estremi dei segmenti, che conosciamo dall'input, ma anche i punti di intersezione che vengono scoperti durante l'esecuzione dell'algoritmo, e che provocano una modifica nell'ordinamento dei segmenti. Verrà testata l'intersezione di due segmenti nello stesso stato di l solo se essi sono adiacenti nell'ordinamento; questo significa che appena un segmento cambia la sua situazione nello stato di l (cioè vi entra –

punto evento=estremo superiore - o viene spostato nell'ordinamento – punto evento=intersezione - o un suo vicino viene eliminato dallo stato – punto evento=estremo inferiore), lo confrontiamo solo con al più altri due segmenti, i suoi adiacenti nell'ordinamento.

Prima di procedere, convinciamoci che, in questo modo, vengono trovate tutte le intersezioni.

Teorema. *Se due segmenti s_i ed s_j intersecano, allora esiste uno stato di l in cui s_i ed s_j sono adiacenti nell'ordinamento.*

Dimostrazione. Poniamoci nel caso generico, in cui cioè i due segmenti siano entrambi non orizzontali e che non vi siano più di due segmenti che si intersecano in uno stesso punto. Tratteremo questi due casi speciali a parte.

Se l'intersezione tra s_i ed s_j si trova sull'estremo di uno dei due segmenti, esso verrà facilmente trovato quando l incontra tale estremo e cambia il suo stato.

Se, invece, l'intersezione è tra due punti interni, supponiamo che l si trovi leggermente sopra il punto di intersezione; se l è abbastanza vicino a tale punto, cioè se non ci sono punti evento tra la posizione di l e il punto di intersezione, allora s_i ed s_j saranno adiacenti nell'ordinamento. D'altra parte, poiché la linea l parte sopra tutti i segmenti, all'inizio il suo stato sarà vuoto, perciò dovrà esistere un momento in cui s_i ed s_j vengono inseriti nello stato ed un momento in cui diverranno vicini nell'ordinamento. Proprio in questo momento la loro intersezione verrà testata ed individuata. Abbiamo quindi dimostrato il caso generale. **CVD**

Studiamo ora i casi particolari che abbiamo escluso precedentemente.

Nel caso in cui uno dei segmenti sia orizzontale, definiamo il suo estremo superiore come quello con ascissa minore, cioè quello più a sinistra, e il suo estremo inferiore come quello con ascissa maggiore, cioè quello più a destra. Con questo accorgimento, questo caso rientra in quello generale.

Se, infine, per una stessa intersezione passano tre o più segmenti, non riscontriamo problemi nel determinare l'intersezione, quanto piuttosto nel descriverla: potremmo richiedere all'algoritmo di dare semplicemente tutti i punti di intersezione una volta, ma è più utile se – per ogni intersezione – esso dia una lista di segmenti che passano attraverso quel punto o via abbiano un estremo.

Osserviamo che questa tecnica garantisce un invariante che rimane vero durante tutto il tempo in cui l spazza il piano: tutti i punti di intersezione che stanno sopra l sono già stati determinati correttamente. Grazie a questo invariante ed al teorema precedente, si deduce che l'approccio è corretto.

Vediamo ora quali siano le strutture dati usate:

- punti evento:
  sono memorizzati in una struttura dati Q detta *coda degli eventi*.
  Dati due eventi p e q, definiamo la relazione d'ordine $\lhd$ come segue: $p \lhd q$ se e solo se o $y(p)<y(q)$ o $y(p)=y(q)$ e $x(p)<x(q)$.
  Q è un albero di ricerca bilanciato, ordinato secondo la relazione d'ordine $\lhd$. Insieme con ogni punto evento p, in Q memorizziamo anche i segmenti che ammettono p come estremo superiore; in questo modo si hanno le informazioni necessarie per gestire l'evento.
  Un'operazione da poter fare su Q consiste nel *rimuovere* il punto evento successivo da Q e restituirlo in modo che possa essere trattato. Tale punto evento è il più alto al di sotto della linea l.
  Un'altra operazione è quella di *inserimento*. Si osservi che due distinti punti evento possono coincidere (ad esempio, gli estremi di due segmenti distinti coincidono, o un'intersezione coincide con un estremo), ma è conveniente trattare tali punti come un unico punto evento, per cui l'operazione di inserimento dovrà essere in grado di verificare se un evento sia già presente in Q.
  Le operazioni di estrazione ed inserimento prendono entrambe tempo $O(\log m)$, dove m è il numero di eventi al momento inseriti in Q, ed m non può mai superare $n+k$, dove k è il

numero di intersezioni.
Si osservi che si è scelto l'albero binario di ricerca e non l'heap per gestire efficientemente il caso in cui ci sono eventi già presenti in Q.

- stato di l:
  può essere memorizzato in un albero binario di ricerca bilanciato in cui i segmenti nello stato risulteranno ordinati visitando l'albero in in-ordine (da sinistra verso destra). Si è scelto un albero binario di ricerca poiché serve una struttura T che mantenga l'ordinamento, che sia dinamica e che, dato un segmento s, sia in grado di ritornare i suoi adiacenti nell'ordinamento.
  Dunque, T deve supportare le operazioni di *inserimento*, *cancellazione,* ricerca del precedente e del successivo. Ogni modifica dell'albero prende $O(\log m')$ tempo, dove m' è il numero di segmenti presenti nello stato di l in un certo istante, ed m' non può mai superare n.

Per concludere, dimostriamo la correttezza dell'algoritmo appena descritto.

Lemma. *L'algoritmo ora descritto determina correttamente tutte le intersezioni e tutti i segmenti che le contengono.*

Dimostrazione. Ricordiamo che la priorità di un evento è data dalla sua ordinata oppure, quando due eventi hanno la stessa ordinata, dalla sua ascissa. Dimostriamo il lemma per induzione sulla priorità dei punti evento.

Sia p un punto d'intersezione ed assumiamo che tutti i punti d'intersezione q con priorità maggiore siano già stati determinati correttamente. Allora, proveremo che anche p ed i segmenti che lo contengono verranno determinati correttamente. Sia $U(p)$ l'insieme dei segmenti che ammettono p come estremo superiore (o sinistro, se orizzontali). Sia $L(p)$ l'insieme dei segmenti che ammettono p come estremo inferiore (o destro, se orizzontali). Sia $C(p)$ l'insieme dei segmenti che ammettono p come punto interno.

Assumiamo prima che p sia l'estremo di qualche segmento. Allora p è memorizzato in Q all'inizio dell'algoritmo, insieme ai segmenti di $U(p)$ e di $L(p)$. I segmenti di $C(p)$ vengono memorizzati in T prima del momento in cui p viene gestito (quando l incontra i loro estremi superiori che, necessariamente giacciono sopra di p), così anch'essi saranno individuati. In questo caso, quindi, il lemma è dimostrato.

Assumiamo ora che p non sia l'estremo di alcun segmento ed, in tal caso, gli unici segmenti coinvolti sono quelli di $C(p)$. L'unica cosa che dobbiamo dimostrare è che p sarà inserito in Q in qualche momento poiché, in tal caso, esso sarà inserito proprio come punto di intersezione. Si considerino tutti i segmenti coinvolti e si immaginino ordinati per angolo che essi formano intorno a p. Siano s_i ed s_j due di tali segmenti e siano essi adiacenti nell'ordinamento appena dato. Allora, c'è un punto evento q con una priorità maggiore di p tale che s_i ed s_j diventano adiacenti quando l oltrepassa q. Per induzione, q è stato gestito correttamente, quindi anche p sarà determinato e memorizzato in Q. **CVD**

Dimostriamo ora che l'algoritmo è output sensitive, e la sua complessità è $O((n+k) \log n)$ dove k è il numero di punti di intersezione.

Lemma. *Il tempo di esecuzione dell'algoritmo ora descritto per un insieme S di n segmenti nel piano è $O(n \log n + k \log k)$, dove k è il numero di intersezioni dei segmenti di S.*

Dimostrazione. L'algoritmo inizia con la costruzione della coda degli eventi Q con gli estremi dei segmenti. Essendo Q un albero binario di ricerca bilanciato, questo passo prende $O(n \log n)$ tempo. Inizializzare T prende tempo costante.

Poi, la retta l comincia a spazzare il piano e vengono gestiti tutti i punti evento, che sono in tutto $2n+k$. Per ciascuno di essi vengono eseguite tre operazioni in Q: l'evento stesso viene cancellato da Q (tempo $O(log\ (n+k))$, viene modificato T a seconda di come è cambiato lo stato di l (tempo $O(log\ n)$), e – se l'ordinamento in T è cambiato – viene verificata l'esistenza di eventuali intersezioni (tempo $O(1)$), in tal caso la nuova intersezione viene inserita in Q (tempo $O(log(n+k))$.

Ne segue che la complessità totale è $O(n\ log\ n) + (n+k)\ O(log\ (n+k))$. Per valutare questa espressione facciamo due casi: *i.* $n \geq k$ e *ii.* $n<k$. Nel primo caso, $k=O(n)$, quindi $O(k\ log\ n)$ viene inglobato in $O(n\ log\ n)$ e la complessità è $O(n\ log\ n)+O(k\ log\ k)=O(n\ log\ n)$. Nel secondo caso, invece, $n=O(k)$ e $O(k\ log\ n)$ viene inglobato in $O(k\ log\ k)$, quindi la complessità diviene $O(n\ log\ n)+O(k\ log\ k)=O(k\ log\ k)$. **CVD**

RIFERIMENTI BIBLIOGRAFICI

[AKS98] P.K. Agarwal, M. van Kreveld, S. Suri: Label Placement by maximum independent set in rectangles. *Computational Geometry: Theory and Applications,* 11 (3-4), 209-218, 1998.

[Bal97] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf: Computational Geometry – Algorithms and Applications. Springer-Verlag, Heidelberg, 1997.

[CLR] Cormen, Leiserson, Rivest: Progetto di algoritmi fondamentali (verificare). N.B. le pagine sono riferite al testo in 3 volumi

[DMal99] S. Doddi, M.V. Marathe, A. Mirzaian, B.M.E. Moret, B. Zhu: Map labelings and its generalizations. *Tech. Rep. LA-UR-96-2411,* Los Alamos National Laboratory. 1999.

[EC96] Edmonson, J. Christensen. A general cartographic labeling algorithm. *Cartographica* Vol.33, N.4, Winter 1996 pp.13-23.

[FW91] M. Formann and F. Wagner: A packing problem with applications to lettering of maps. *Proc. 7th Annual Symp. On Comp. Geom (SCP '91)*, 281-288, 1991.

[IL97] C. Iturriaga, A. Lubiw: NP-hardness of some map labeling problems. *Tech. Rep. CS-97-18,* University of Waterloo. 1997.

[KI88] T. Kato, H. Imai: The NP-Completeness of the character placement problem of 2 or 3 degrees of freedom. *Record of Joint Conf. Of electrical and Electronic Engineers in Kyushu, p. 1138, 1998.*

[KW98] M. Kaufmann, D. Wagner (Eds.): *Drawing Graphs – Methods and Models*. Lecture Notes in Computer Science 2025, Springer 1998.

[KT98] K.G. Kakoulis, I.G. Tollis: A unified approach to labeling graphical features. *Proc. 14th Annual ACM Symp. on Computational Geometry (SCG'98)*, pp. 347-356, 1998.

[KSW98] M. van Kreveld, T. Strijk, A. Wolff: Point set labeling with sliding labels. *Proc. 14th Annual ACM Symp. on Computational Geometry (SCG'98)*, pp. 337-346, 1998.

[MS91] J. Marks, S. Shieber: The computational complexity of cartographic label placement. Tech. Rep. TR-05-91 Harvard Univ. Comp. Sci. 1991.

[MC80] C.Mead and L.Conway, *Introduction to VLSI Systems*, Addison Wesley, 1980.

[P94] C.H. Papadimitriou: Computational complexity, Addison-Wesley Publ. Co. 1994.

[W94] F. Wagner: Approximate map labeling is in Omega(*n log n). Inf. Proc. Letters,* 52(3), pp. 161-165, 1994.