Note del Corso di Modelli Biologici Discreti: Un paio di algoritmi DNA per risolvere SAT

Giuditta Franco

Corso di Laurea in Bioinformatica - AA 2012/2013

Uno dei più grossi risultati nell'informatica degli ultimi vent'anni è stato capire che molti importanti problemi di ricerca computazionale e di interesse pratico sono NP-completi, e quindi in generale privi di algoritmi efficienti che li risolvano esattamente; infatti, tali algoritmi hanno un costo esponenziale con le dimensioni del problema, e quindi sono utili solo per istanze di piccole dimensioni. Negli ultimi decenni questi problemi *intrinsecamente complessi* sono stati studiati con molta attenzione: se per qualche problema NP-completo fossimo riusciti a trovare un metodo esatto con costo polinomiale, questo si sarebbe potuto tradurre in una procedura per risolvere tutti i problemi della classe NP, ma a tutt'oggi una tale risolubilità non è stata ottenuta e si è propensi a credere che non sia ottenibile.

Visto l'interesse che ruota attorno ai problemi NP-completi, si sono sviluppati metodi euristici con costi polinomiali (proporzionali a piccole potenze della dimensione del problema), che nella pratica lavorano bene, nonostante non garantiscano l'ottimalità della soluzione. Infatti, con l'aumentare delle dimensioni del problema, l'analisi del caso peggiore diventa sempre più irrilevante (perché più improbabile), e il comportamento medio dell'algoritmo domina l'analisi delle applicazioni pratiche.

Le euristiche sono specifiche per il problema, nel senso che non è garantito che una procedura euristica che trova una quasi-soluzione per un problema NP-completo sia valida anche per un altro problema. Negli ultimi anni sono stati dedicati molti sforzi allo studio di un classico problema NP-completo di ottimizzazione combinatoria, che è quello del *commesso viaggiatore*: date N città e un criterio per calcolare il costo del viaggio tra due di queste, si chiede qual è il percorso di costo minimo tra quelli che partono da una città e passando da ogni altra una sola volta alla fine tornano nella città di partenza. Il più grosso problema di commesso viaggiatore di cui si è calcolata e pubblicata una soluzione esatta ha 318 città.

In questo contesto è rilevante il modello di calcolo probabilistico, in cui però non si ha la certezza della correttezza del risultato, sebbene si tratti di una correttezza nota in senso probabilistico, come nel caso dell'algoritmo di tipo Montecarlo. Sono significativi anche gli algoritmi Las Vegas, che non sempre forniscono un risultato, ma quando lo forniscono questo è certamente corretto. La probabilità di insuccesso in tutte le applicazioni di un tale algoritmo decresce esponenzialmente a 0 con il numero delle applicazioni, che sono tra di loro indipendenti; ne segue che è un algoritmo che si adatta bene al calcolo parallelo (ad esempio si può usare per verificare la correttezza dei risultati forniti da un algoritmo Montecarlo).

In generale, per i problemi NP-completi viene molto utilizzato il modello di *calcolo approssimato*, che ammette risultati affetti da un errore controllabile a priori. Gli algoritmi di approssimazione garantiscono una soluzione dentro qualche percentuale di ottimalità: per esempio se ne conoscono di efficienti per SAT, in quanto forniscono un assegnamento di valori di verità che soddisfa un numero di clausole che è almeno il 75% del numero di clausole soddisfatte dal miglior possibile assegnamento. E in molti campi, che vanno dall'analisi combinatoria classica alla fisica statistica, c'è un interesse crescente nell'usare SAT come formulazione di riferimento dei problemi di interesse pratico. In questo contesto di algoritmi non esatti ma "efficienti" per i problemi NP-completi trovano ampio inserimento le proposte del DNA Computing.

Ricordiamo che "risolvere SAT" vuol dire trovare, se esiste, un'assegnamento di valori di verità per le variabili $x_1, x_2, \ldots, x_n$ della formula proposizionale data ϕ, che la rendono vera. Generalmente esprimiamo ϕ come la congiunzione di m clausole $C_1, C_2, \ldots, C_m$, e cerchiamo un assegnamento che *soddisfi* tutte le m clausole (da qui il nome di *soddisfacibilità proposizionale* o SAT). Nel risolvere SAT non si perde in generalità prendendo in considerazione un 3-SAT (istanza in cui ogni clausola contiene al più tre letterali).

Nel seguito indicheremo sempre con n il numero di variabili della data ϕ e con m il numero di clausole di cui ϕ è congiunzione. Inoltre, denoteremo con *Lit(C)* l'insieme dei letterali che occorrono nella clausola C, e con *Cla(L)* l'insieme degli indici delle clausole in cui occorre il letterale L.

1 Connettività di grafi

Nel 1995 Richard J. Lipton ha generalizzato l'esperimento di Adleman del 1994, proponendo il seguente algoritmo DNA per risolvere SAT.

1. Si codificano tutti i possibili assegnamenti (2^n) con una gran quantità di stringhe di DNA, tali che, per ogni variabile proposizionale x_i con $i = 1, \ldots, n$, un oligonucleotide che codifica o x_i o $\neg x_i$ occorre nella stringa.

2. Presi tutti gli assegnamenti, per ciascuna clausola si effettua un filtraggio guidato da procedure di separazione, che prende le stringhe ove occorre almeno uno dei letterali della clausola.

   Quindi, partendo dalle stringhe iniziali che codificano gli assegnamenti, si selezionano quelle che soddisfano una prima clausola, delle stringhe risultanti si selezionano quelle che soddisfano una seconda clausola, e così via fino alla selezione per la $m-$ esima clausola. Se dopo queste separazioni rimane qualche stringa di DNA, il corrispondente assegnamento rende vere tutte le clausole di ϕ e risolve SAT.
   Più precisamente, se stiamo facendo la selezione per la $j-$ esima clausola, e per esempio $C_j = \neg x_3 \vee x_6 \vee x_8$, si procede separando le stringhe ottenute fino a quel punto in due provette A e B, la prima contenente le stringhe in cui occorre l'oligonucleotide che codifica $\neg x_3$, e la seconda contenente le stringhe in cui $\neg x_3$ non occorre. Le stringhe di B vengono poi separate in due provette C e D: in C abbiamo le stringhe in cui occorre x_6 e in D quelle in cui x_6 non occorre. Infine si separano le stringhe di D nelle provette E ed F, tali che la prima contiene le stringhe in cui occorre x_8 e la seconda le rimanenti. È chiaro che $A \cup C \cup E$ contiene tutte le stringhe che rendono vera C_j ed F tutte le altre. Nel prossimo filtraggio si prendono in considerazione solo le stringhe della provetta $A \cup C \cup E$.

Lipton adotta chiaramente il metodo della forza bruta nel generare tutti gli assegnamenti e poi passare alle separazioni successive, e questo comporta una quantità di DNA non proponibile nel caso di istanze di grosse dimensioni. Inoltre, il numero di separazioni richieste è lineare in m, nel caso di un problema 3-SAT (n, m) esatto è $3m$.

Per quanto riguarda l'implementazione col DNA, l'algoritmo di Lipton nel primo passo utilizza la codifica Adleman, in quanto alla generazione di tutti i possibili assegnamenti (che possiamo vedere come un numero composto da n cifre binarie) iniziali fa corrispondere la generazione di tutti i possibili cammini di un grafo G_n associato all'istanza ϕ.

Definizione 1. *G_n ha i nodi $a_1, x_1, x_1', a_2, x_2, x_2', \ldots, x_n, x_n', a_{n+1}$ con gli archi orientati nel modo seguente: da a_k sia verso x_k che verso x_k', e sia da x_k che da x_k' verso a_{k+1}, con $k = 1, 2, \ldots, n$.*

Il grafo G_n è costruito in modo tale che, tutti i cammini che partono da a_1 e arrivano ad a_{n+1} codificano un numero binario di n cifre (e quindi uno degli assegnamenti): perché su ogni nodo a_i si deve scegliere tra la variabile x_i e la sua negata, e fatta questa scelta il percorso fino ad a_{k+1} è obbligato. Secondo la codifica di Adleman ad ogni cammino legale di G_n corrisponde una sequenza correttamente concatenata sia di vertici che di archi. Inoltre, il grafo di Lipton ha un'importante proprietà di simmetria: tutti i cammini hanno la stessa probabilità di essere generati.

Il secondo passo dell'algoritmo è stato implementato, in modo rozzo rispetto ai nostri giorni, con una tecnica di separazione che sfrutta l'affinità biotina-avidina.

Lipton risolve una semplicissima istanza per 3-SAT(2,2) che andiamo ad utilizzare come esempio esplicativo: si tratta della formula

$$G_1 = (x \vee y) \wedge (x' \vee y')$$

dove x' è il negato di x e y' il negato di y. Vengono generati gli assegnamenti 00, 01, 10, 11, come cammini del grafo in Figura 1.

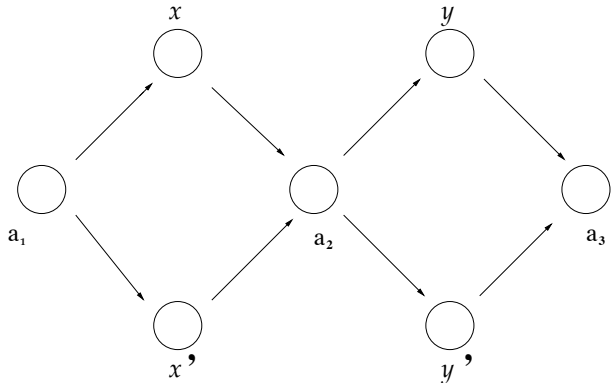

Figura 1: Grafo relativo alla formula G_1

Tali assegnamenti, secondo l'algoritmo di Lipton, vengono processati come indica la Tabella 1 (ove T è la provetta input):

2 Connettività di clausole

Nel '99 Natasa Jonoska et al. riprendono l'idea del *Contact Network* introdotta da Lipton, e la realizzano teoricamente proponendo la costruzione di un grafo tridimensionale che riduce notevolmente il tempo e le operazioni necessarie per identificare una soluzione di 3-SAT.

provette	assegnamenti presenti
T	00, 01, 10, 11
A	10, 11
B	00, 01
C	01
A ∪ C = T ′	10, 11, 01
A ′	01
B ′	10, 11
C ′	10
A ′ ∪ C ′	01, 10

Tabella 1: Passi effettuati dall'algoritmo per la formula G_1

Definizione 2. *Chiamiamo* Contact Network *un grafo orientato con un singolo nodo di partenza s e un singolo nodo d'arrivo t fissati. Ciascun arco è etichettato con x o* $\overline{x}$ *(la negata di x), dove x è una variabile. Dato un assegnamento di valori alle variabili, un arco è considerato* connesso *se la variabile su tale arco vale* 1 *(quindi, se l'arco è etichettato con* $\overline{x}$ *allora è connesso quando x = 0).*

Si può definire il Contact Network di una generica formula con una sorta di induzione che traduce la disgiunzione tra formule nella disposizione in parallelo dei grafi corrispondenti, e la congiunzione di formule nei rispettivi grafi disposti in serie (vedi Figura 2).

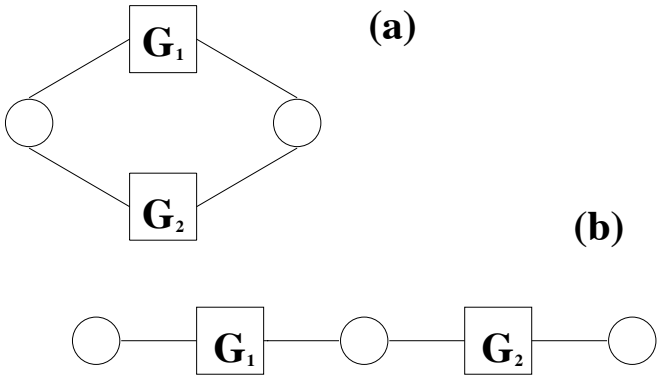

Figura 2: Contact network di (a) $\alpha = \beta \vee \gamma$ e di (b) $\alpha = \beta \wedge \gamma$, dove G_1 è il contact network di β e G_2 di γ

Un problema di Contact Network consiste nel determinare se esiste o meno un assegnamento di valori delle variabili (che etichettano gli archi) tale che vi sia un cammino orientato e connesso (composto da archi connessi) che porta da s a t.

Un problema SAT è equivalente al problema del Contact Network associato, che è strutturato come segue: per ogni clausola della formula vi sono due nodi collegati tra di loro con tanti archi (ciacuno etichettato con un letterale) quanti i letterali della clausola, tali strutture relative ad ogni clausola sono disposte in serie tra il nodo s e il nodo t andando a formare il grafo che rappresenta la formula.

Per esempio, se consideriamo come istanza per un 3-SAT(3,3) la formula

$$G_2 = (x \vee y) \wedge (\overline{x} \vee y \vee z) \wedge (\overline{x} \vee y \vee \overline{z})$$

ove i simboli barrati indicano le negate delle variabili, il corrispondente grafo è quello in Figura 3.

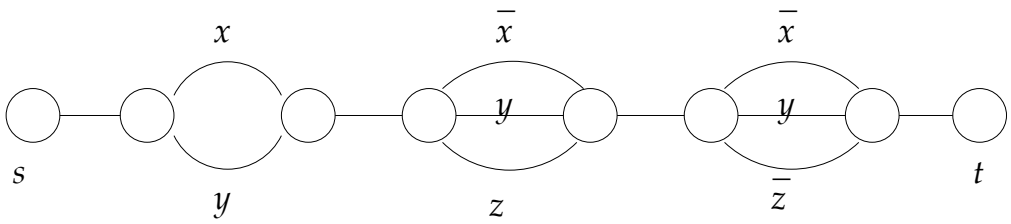

Figura 3: Contact network associato alla formula G_2

Viene dunque proposto il seguente algoritmo: si consideri un multinsieme con molte copie di un grafo G, che è un'istanza del problema Contact Network, e per ogni variabile x_i, con $i = 1, 2, \ldots, n$, si ripeta il seguente ciclo di operazioni:

1. si separi il multinsieme in due parti A e B, ciascuna contenente circa metà delle stringhe,
2. in A si eliminino dai grafi G gli archi etichettati con x_i e in B quelli etichettati con $\overline{x_i}$, ottenendo così A' e B',
3. si uniscano A' e B' ($A' \cup B'$ è dunque il multinsieme su cui viene applicato il ciclo successivo).

La formula proposizionale associata al grafo G è soddisfacibile se, alla fine di questo processo, rimane nel multinsieme un grafo che connette il nodo di partenza s col nodo d'arrivo t, e questo si formalizza nella seguente

Proposizione 3. *Sia A un multinsieme di molecole che, dopo l'effettuazione dei passi 1,2,3 della procedura appena vista, contengono entrambe le molecole codificanti s e t. Allora $A \neq \emptyset$ se e solo se c'è una soluzione al problema 3-SAT.*

Dimostrazione. Sia ϕ una formula logica associata al Contact Network. Supponiamo $A \neq \emptyset$, ovvero che c'è una molecola che contiene s e t. Questo significa che, dentro il grafo, per ogni coppia di vertici adiacenti che codificano una clausola c'è un arco che li collega. Tale arco non è stato rimosso nel passo 2, vuol dire che è un arco x (letterale della clausola corrispondente alla coppia di vertici) che prende valore 1. Ma il passo 2 assicura che la variabile x prende valore 1 (non viene eliminata) se e solo se $\overline{x}$ prende valore 0 (viene eliminata), e viceversa. Quindi, l'esistenza di una molecola che collega il nodo di partenza con quello d'arrivo implica l'esistenza di un assegnamento consistente di 0 e 1 alle variabili che rende vera la formula ϕ; se ne deduce che il problema 3-SAT ha soluzione.

Viceversa, se 3-SAT ha soluzione allora ci sono valori per le variabili che rendono vera ciascuna clausola. E quindi due vertici adiacenti codificanti una clausola nel corrispondente Contact Network sono collegati da almeno un arco, etichettato con il letterale che vale 1. Pertanto, in corrispondenza del valore 1 del letterale, l'arco corrispondente non viene tagliato, e i due vertici risultano connessi. Quindi s e t risultano connessi: ne segue che $A \neq \emptyset$. □

È importante notare che nessuno dei passi dell'algoritmo dipende dal numero di vertici o di archi del grafo. L'algoritmo dipende solo dal numero di variabili nella formula.

Per quanto riguarda l'implementazione, viene suggerita la costruzuone dei grafi del problema contact network con strutture tridimensionali di DNA. Questo tipo di approccio è stato ispirato da diversi lavori di Seeman et al. che, nell'arco di dieci anni ('89 -'99), hanno dimostrato che è possibile costruire in laboratorio particolari strutture tridimensionali (vedi Figura 4) con molecole di DNA, e che quando tali strutture hanno al più grado 4 (ovvero codificano vertici del grafo collegati a non più di 4 archi) risultano stabili e riproducibili in gran quantità.

L'implementazione DNA cosiste nei passi seguenti.

1. Si combinino in una provetta molte copie di molecole DNA codificanti i vertici, gli archi, il nodo di partenza, e quello di arrivo; si provochi un annealing che grazie alla complementarietà studiata nella codifica porta alla formazione dei grafi tridimensionali, tipo quello della Figura 4.

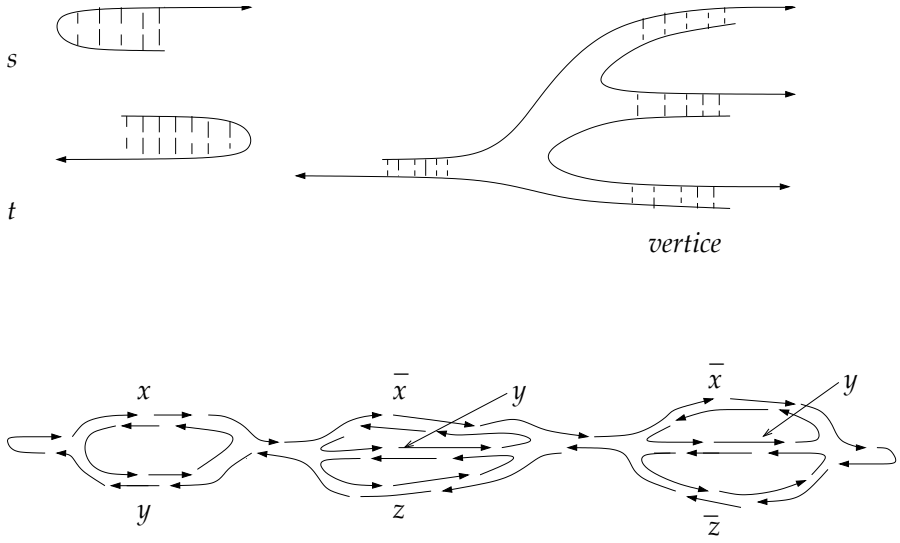

Figura 4: Molecole codificanti la s, la t, e i vertici del grafo relativo a G_2

2. Si eliminino i grafi che non si sono formati completamente (quelli che hanno delle parti terminali non appaiate), per esempio con un enzima exonucleasi.

3. Per ciascuna variabile nella formula si ripeta il seguente ciclo:

   (a) si divida la soluzione in due provette 1 e 2,

   (b) in 1 si aggiunga l'enzima che taglia l'oligonucleotide che codifica la variabile x (scelto appositamente con quel sito di restrizione), in 2 quello che taglia l'oligonucleotide che codifica $\neg x$,

   (c) si aggiungano nella soluzione delle strutture della forma delle molecole codificanti s e t (vedi Figura 4), ma fatte in modo che vadano a "chiudere" i tagli fatti dagli enzimi (vedi punto (c) della Figura 5), si facciano ibridizzare e si effettui la ligation,

   (d) si uniscano le provette 1 e 2;

4. si amplifichino i grafi "buoni" con cicli di PCR fatti utilizzando come primers il complementare del nodo di partenza e il nodo d'arrivo.

In linea di principio il modello DNA plastico proposto è perfettamente realizzabile. Tuttavia, la costruzione di tali strutture richiede un tipo di biotecnologia molto avanzata e una grande sperimentazione nel determinare le

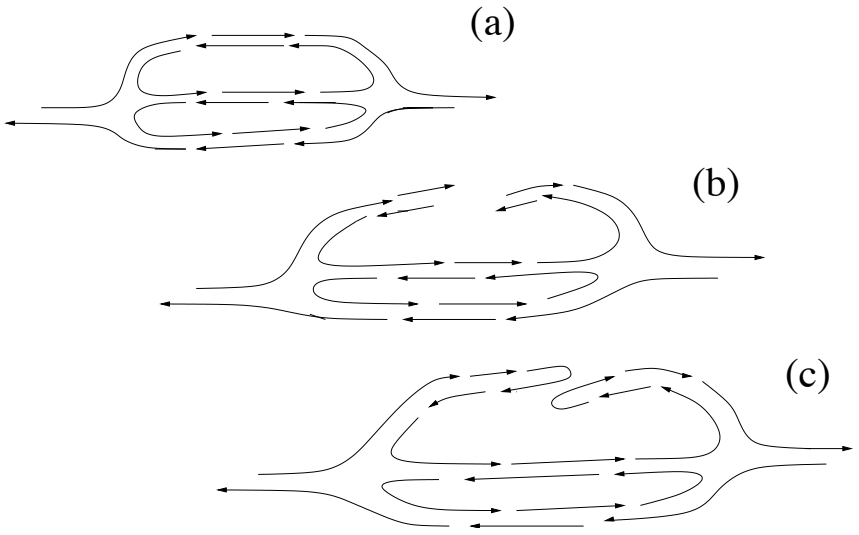

Figura 5: I tagli enzimatici nell'algoritmo DNA

condizioni adatte alle reazioni. E anche quando si sono determinate tali condizioni, non è banale applicare le tecniche standard di biologia molecolare alle strutture codificanti i grafi. Inoltre, solo una piccola frazione del DNA utilizzato riesce a formare tali strutture, e questo limita notevolmente il numero di tagli enzimatici che si possono effettuare.

Mentre nell'algoritmo proposto da Lipton si viene a formare un numero esponenziale di molecole distinte, codificanti tutti i cammini del grafo da s a t, e poi viene estratta la soluzione, qui si forma un unico grafo che contiene tutti i possibili cammini. Tuttavia, siccome il numero dei grafi viene diviso per 2 ad ogni applicazione del ciclo dell'algoritmo, abbiamo bisogno di almeno 2^n copie del grafo e quindi, costruire un numero sufficiente di tali strutture richiede una quantità esponenziale della massa di DNA necessaria per determinare la soluzione del problema.

Si è stimato che ogni provetta può contenere circa 10^{15} molecole di DNA, e di queste 10^9 possono essere distinte. Le molecole di DNA codificanti un vertice di grado 4 (come quello rappresentato nella Figura 4) richiedono almeno 200 nucleotidi, quindi sembra appropriato contare, per ciascuno di tali vertici, 4 molecole di DNA. Per una formula 3-SAT con m clausole abbiamo bisogno di $8m$ molecole per i vertici, $3m$ per gli archi, $6m$ strutture per "tappare" i tagli enzimatici, e 2 molecole codificanti s e t, ovvero abbiamo bisogno di $17m+2$ molecole distinte. Considerando che necessitano 2^n copie

del grafo DNA, abbiamo un limite alle dimensioni del nostro problema dovuto alla quantità di DNA: $(17m + 2)2^n < 10^9$. Nei problemi reali m è almeno 3 volte più grande di n, quindi con questo algoritmo non sarebbe possibile affrontare nemmeno un problema con 20 variabili.

Inoltre, l'algoritmo richiede la disponibilità di $2n$ enzimi di restrizione che agiscano in tempi, condizioni ed efficienze paragonabili.

Riduzione della complessità

Per migliorare l'efficienza degli algoritmi DNA si possono seguire tre strade (o trovare un buon equilibrio tra i miglioramenti in una direzione e gli eventuali peggioramenti nelle altre):

1. Diminuire la dimensione dello spazio delle soluzioni.
2. Trovare implementazioni più efficienti dell'operazione separazione.
3. Diminuire il numero di separazioni.