Scacchi, parenti e serpenti

Andrea Casolaro

Il problema che Rudy deve affrontare può essere modellizzato considerando i parenti invitati al pranzo di ferragosto come i vertici di un grafo non orientato i cui archi uniscono i parenti che non nutrono antipatie reciproche.

In tale modello, il problema di trovare la disposizione dei commensali in modo che ciascun parente sia seduto vicino due con cui è in sintonia è equivalente a trovare un cosiddetto "ciclo Hamiltoniano" nel grafo degli invitati corrispondente. Un ciclo Hamiltoniano è infatti un percorso chiuso che passa per ciascuno dei vertici di un grafo esattamente una volta.

Prima di capire come trovare questo percorso è però necessario capire se possa innanzitutto esistere con le premesse fornite dal problema. Nella teoria dei grafi, il teorema di Dirac fornisce una condizione sufficiente affinchè un grafo con un numero di vertici maggiore di 3 (nel nostro problema il caso $n = 1$ è banale) sia Hamiltoniano, ovvero in grado di contenere un ciclo Hamiltoniano. Indicando con p un generico parente-vertice del nostro grafo, la suddetta condizione è che il grado di p, ovvero il numero di archi che connettono p al resto del grafo, sia maggiore o uguale alla metà del numero dei vertici, ovvero il numero di invitati al pranzo:

$$deg(p) >= (2n)/2 = n$$

Nel nostro modello, $deg(p)$ corrisponde al numero di parenti con cui p è in sintonia. Sappiamo che a ciascun commensale stanno antipatici fino a $n - 1$ altri commensali. In tal caso avremo quindi che:

$$(2n - 1) - (n - 1) = n <= deg(p) <= 2n - 1$$

Con le premesse di Rudy la condizione del teorema di Dirac è quindi rispettata per qualsiasi vertice del grafo ed esiste dunque almeno una disposizione dei commensali per assicurarsi un tranquillo ferragosto. Trovare un ciclo Hamiltoniano in un grafo generico è però un problema computazionalmente complesso facente parte della cosiddetta classe dei problemi NP-completi. Un approccio brute-force in cui provare una qualsiasi combinazione dei commensali a tavola è chiaramente inefficiente in quanto esistono $(2n)!$ combinazioni diverse. Fortunatamente, esiste un algoritmo, ideato da Edgar Palmer nel 1997, che è in grado di trovare un ciclo Hamiltoniano in un modo semplice ed intuitivo, purchè il grafo soddisfi il teorema di Ore, una generalizzazione del teorema di Dirac.

Per illustrare questo algoritmo consideriamo la Fig.1 in cui ciascun parente è rappresentato da un pezzo degli scacchi. L'idea è quella di partire da una disposizione qualsiasi (anche randomica) dei commensali e inziare a considerare in modo ordinato le coppie di parenti seduti vicino e vedere se tra di loro c'è antipatia o meno. Se non vi è antipatia si considera la coppia successiva, in caso contrario occorre spostare uno dei due parenti. Con riferimento alla Fig.1(a), supponiamo che tra il parente p_i e il parente p_{i+1} vi sia un'antipatia. In questo caso si procede considerando in ordine ciascun parente a partire da p_{i+2} fino ad arrivare al parente p_j tale che tra p_i e p_j e tra p_{i+1} e p_{j+1} non vi siano antipatie. A questo punto si effettua una inversione della disposizione tra p_{i+1} e p_j (inclusi) e si ottiene la disposizione mostrata in Fig.1(b). A partire da quest'ultima si prosegue dal parente p_j come all'inizio e ad ogni nuova coppia di parenti che non possono star seduti vicino si effettua lo "swap" descritto precedentemente fino a quando ogni parente non è seduto vicino due con cui è in sintonia.

Per convincersi della correttezza dell'algoritmo consideriamo ancora una volta le disposizioni mostrate in Fig.1 prima e dopo lo swap. Nel primo caso sappiamo che esiste un'antipatia, gap nella terminologia formale, tra p_i e p_{i+1} ma non sappiamo nulla circa le eventuali antipatie tra i commensali tra p_{i+1} e p_{j+1}. Nel secondo caso, invece, non abbiamo antipatie tra p_i e p_j e tra p_{i+1} e p_{j+1} ma ancora non possiamo dire nulla circa i commensali tra p_j e p_{i+1}. Tuttavia, poichè l'antiapatia è appunto reciproca, commutativa, notiamo che le eventuali antipatie presenti tra i commensali tra p_{i+1} e p_j sono preservate dallo swap. Ne segue che, indipendetemente dal rapporto

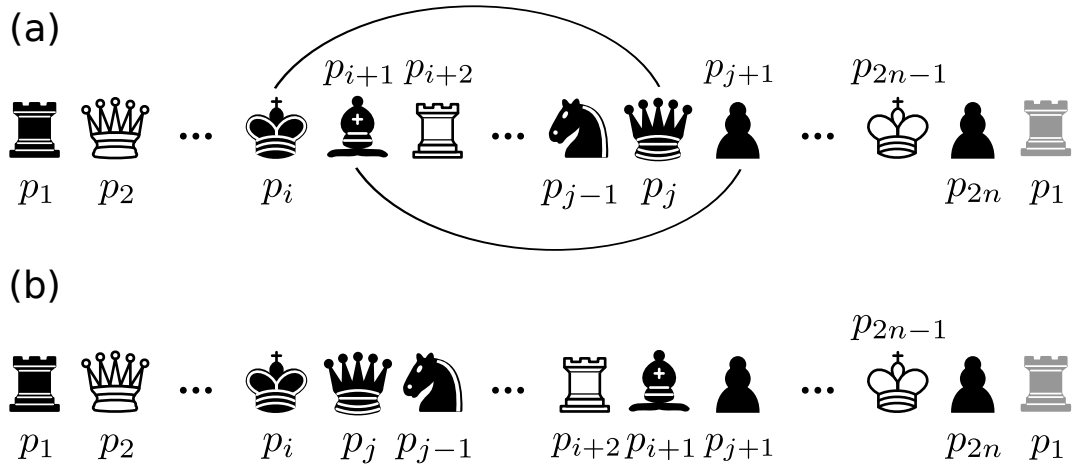

Figura 1: (a) Disposizione prima dello swap. (b) Disposizione dopo lo swap.

tra p_j e p_{j+1} prima dello swap, la disposizione in Fig.1(b) ha almeno un'antipatia in meno tra i commensali rispetto a quella in Fig.1(a). Di conseguenza, per trovare la disposizione finale, nel caso peggiore possibile, sarà necessario effettuare un numero di scambi tra parenti al più pari al numero di invitati al pranzo, ovvero $2n$.

Prima di proseguire con il calcolo formale della complessità di tale algoritmo è necessario però precisare che l'esistenza del parente p_j nella procedura sopracitata è proprio assicurata dal fatto che il grafo soddisfa il teorema di Dirac. Dal momento che quest'ultimo fornisce una condizione sufficiente ma non necessaria affinchè un grafo sia Hamiltoniano, ne segue che tale algoritmo è applicabile solo ad una sottoclasse di grafi Hamiltoniani mentre il caso generale rimane tuttora un problema di estrema complessità.

Indicando con $G = (V, E)$ il grafo degli invitati al pranzo con $|V| = 2n$ parenti e $|E|$ archi che connettono i parenti in sintonia, possiamo formalizzare l'algoritmo esposto con il pseudo-codice mostrato in fondo alla pagina dove si è usata per comodità la notazione $x^* = x \bmod |V|$. Dalle considerazioni fatte precedentemente è chiaro che il ciclo 4-10 è eseguito al più $|V|$ volte e, se il grafo è implementato attraverso una matrice delle adiacenze, le righe 5 e 7 sono eseguite in $O(1)$. Poichè la complessità di *ReverseCycle* è $O(|V|)$, il ciclo 6-9 è dunque eseguito in $O(|V|)$. In definitiva quindi l'algoritmo ha una complessità totale pari a $O(|V|^2) = O(n^2)$, estremamente più bassa della complessità esponenziale di un'approccio brute-force.

Possiamo dunque concludere che, anche per un numero moderatamente grande di invitati al pranzo di ferragosto, Rudy e Gaetanagnesi possono facilmente trovare la disposizione corretta dei commensali con una procedura semplice e applicabile anche manualmente (anzi zampalmente) con i pezzi degli scacchi!

Algorithm 1 Criss-Cross Algorithm from Palmer, 1997

```
1:  procedure CrissCross(G)
2:      Π ← {v | v ∈ V[G]}
3:      i ← 0
4:      while there are gaps do
5:          if (Π[i*], Π[(i + 1)*]) ∉ E[G] then
6:              for j ← 2 to |V| − 2 do
7:                  if (Π[i*], Π[(i + j)*]) ∈ E[G] and (Π[(i + 1)*], Π[(i + j + 1)*]) ∈ E[G] then
8:                      ReverseCycle(Π,(i + 1)*,(i + j)*)
9:                      break
10:         i ← i + 1
11:     return Π
```