5. Problemi di Ottimizzazione e Programmazione Matematica

Marco Liverani

Università degli Studi Roma Tre
Dipartimento di Matematica e Fisica
Corso di Laurea in Matematica
E-mail liverani@mat.uniroma3.it

Aprile 2014

Revisione del 13 aprile 2014.

1 Generalità

In queste pagine ci proponiamo di descrivere una particolare classe di problemi, quelli di ottimizzazione combinatoria, studiandone una formulazione più generale che si inquadra nel contesto più ampio di problemi di *programmazione matematica*. Per iniziare a circoscrivere la famiglia dei problemi di ottimizzazione combinatoria e le caratteristiche che la distinguono da altri insiemi di problemi, dobbiamo focalizzare l'attenzione sul tipo di soluzione che viene richiesta da ciascuna classe di problema: è proprio la tipologia della soluzione che si deve individuare per risolvere il problema che ci permette di costruire una prima classificazione che aggrega in famiglie omogenee problemi che per ambito, applicazioni, strumenti di risoluzione risultano altrimenti molto diversi fra loro. Nel costruire questa classificazione procederemo in modo "incrementale", rendendo la richiesta posta dal problema stesso, via via sempre più "esigente".

La formulazione più essenziale in cui può essere posto un problema è quella in cui si chiede semplicemente di rispondere «sì o no», «vero o falso», «0 o 1». Esiste uno zero della funzione $f(x)$ nell'intervallo (a,b)? Esiste un cammino di lunghezza inferiore a k tra i due vertici u e v del grafo G? È possibile trovare un cammino chiuso sul grafo per raggiungere ogni vertice senza passare mai due volte per lo stesso spigolo? È possibile vincere una determinata partita a scacchi in sole tre mosse partendo da una certa configurazione dei pezzi sulla scacchiera? Tutti problemi la cui soluzione è effettivamente un semplice «sì o no», «esiste o non esiste», «è possibile o non è possibile». Non si richiede infatti di costruire ed esibire uno zero della funzione $f(x)$, un cammino sul grafo G, o la sequenza di mosse che mi permette di vincere la partita a scacchi, ma soltanto di dire se uno zero, un cammino o una sequenza di mosse vincenti *esiste*. Questo genere di problemi sono denominati **problemi di decisione** o **di esistenza**. È bene precisare che non solo non è richiesto di esibire una possibile configurazione delle variabili del problema a supporto della soluzione del problema stesso, ma spesso non è neanche necessario costruire la soluzione per poter formulare una risposta per una particolare istanza di un problema di decisione: è sufficiente articolare la risposta in modo tale che, al di là di ogni dubbio, questa risulti inconfutabile. Ad esempio, se la funzione $f(x)$ è continua nell'intervallo (a,b) e risulta $f(a) > 0$ e $f(b) < 0$ allora certamente esiste un $\tilde{x} \in (a,b)$ tale che $f(\tilde{x}) = 0$. Al contrario se il grafo G non è connesso e i due vertici u e v appartengono a due componenti connesse distinte, allora certamente un cammino $p : u \rightsquigarrow v$ non esiste in G.

Nei cosiddetti **problemi di ricerca**, invece, viene richiesto proprio di costruire una soluzione del problema, ossia si richiede di esibire esplicitamente una configurazione delle variabili del problema che diano luogo ad una soluzione. Rifacendoci agli esempi formulati in precedenza, si richiede di identificare un particolare valore di $x \in (a,b)$ tale che $f(x) = 0$, oppure di costruire uno specifico cammino $p : u \rightsquigarrow v$ sul grafo G con meno di k spigoli, o un circuito su G che non passi mai due volte per lo stesso spigolo, o la sequenza di mosse che mi permetta di vincere la partita a scacchi a partire da una certa disposizione dei pezzi sulla scacchiera.

È chiaro che una soluzione può essere esibita solo se esiste: in altre parole risolvere un'istanza di un problema nella sua formulazione come *problema di ricerca*, comporta implicitamente anche la soluzione dello stesso problema nella sua formulazione come *problema di decisione*.

Se viene richiesto di calcolare ed esibire *tutte* le possibili soluzioni di una determinata istanza del problema, allora siamo di fronte ad una formulazione del problema come **problema di enumerazione**. Ad esempio si richiede di identificare tutti i valori $x \in (a,b)$ tali che $f(x) = 0$; si richiede di costruire tutti i possibili cammini sul grafo G da u a v composti con meno di k spigoli; ecc. Anche in questo caso dovrebbe risultare chiaro che risolvere un problema di enumerazione significa, implicitamente, risolvere anche un problema di ricerca, in cui si chiede di esibire *una sola* soluzione ammissibile per il problema, ed anche risolvere il problema di decisione associato, in cui si chiedeva soltanto di stabilire

Classe	Descrizione
Problemi di decisione o di esistenza	Si richiede di verificare se *esiste* o meno una certa soluzione per l'istanza del problema
Problemi di ricerca	Si richiede di costruire ed esibire *una* soluzione per l'istanza del problema
Problemi di enumerazione	Si richiede di costruire ed esibire *tutte* le soluzioni per l'istanza del problema
Problemi di ottimizzazione	Si richiede di costruire le soluzioni *ottimali* in base ad un determinato criterio (funzione obiettivo o di costo)

Tabella 1: Una classificazione dei problemi in base al tipo di soluzione richiesta

se una soluzione esiste o meno per quella determinata istanza del problema. Risolvere problemi di enumerazione sarà quindi più laborioso che risolvere problemi di ricerca; risolvere problemi di ricerca sarà più laborioso che risolvere problemi di esistenza.

Arriviamo così alla formulazione dei **problemi di ottimizzazione**: in questo caso non ci si accontenterà né di stabilire soltanto se una soluzione esiste, né di identificare una soluzione qualsiasi per l'istanza del problema e neanche di identificare tutte le soluzioni del problema stesso, indistintamente; si richiede, invece, di identificare una o tutte le soluzioni che risultano ottimali rispetto ad un qualche criterio di valutazione delle soluzioni ammissibili per una specifica istanza del problema. Ad esempio tra tutti gli zeri della funzione $f(x)$ nell'intervallo (a,b) si chiederà di individuare quelli che rendono minima la funzione $\varphi(x)$ nell'intervallo stesso; tra tutti i cammini che sul grafo G collegano i vertici u e v si chiederà di trovare quelli di lunghezza minima o che minimizzano il costo dato dalla somma dei pesi attribuiti a ciascuno spigolo del grafo, e così via.

In un problema di ottimizzazione viene quindi fornita anche una funzione con cui valutare il costo di ciascuna soluzione del problema: questa funzione, detta *funzione obiettivo*, costituisce un criterio di scelta tra le possibili soluzioni del problema, ossia uno strumento, una bussola, con cui orientarsi nell'insieme delle soluzioni ammissibili con l'intento di selezionare solo le soluzioni migliori, quelle che "minimizzano" il costo o massimizzano il "profitto".

Nel seguito ci occuperemo quasi esclusivamente di problemi di ottimizzazione; in particolare, come vedremo tra breve, ci occuperemo di una particolare classe di problemi che va sotto il nome di problemi di *ottimizzazione combinatoria*. In generale per questi problemi l'insieme delle soluzioni ammissibili entro cui cercare il sottoinsieme di soluzioni ottimali è un insieme discreto, di cardinalità finita; tuttavia il numero di elementi di cui è costituito è di un ordine di grandezza molto superiore alla dimensione dell'istanza del problema che si intende risolvere.

2 Formulazione generale per i problemi di ottimizzazione

In termini molto generali un'istanza di un problema di ottimizzazione è definita come una coppia (A,f), dove A è l'**insieme delle soluzioni ammissibili** ed $f : A \to \mathbb{R}$ è la **funzione obiettivo** che si deve ottimizzare (massimizzare o minimizzare). Il problema di ottimizzazione viene posto chiedendo di trovare i valori $x \in A$ tali che $f(x) \leq f(y)$ per ogni $y \in A$ (problema in forma di minimizzazione). Tali valori x costituiscono delle **soluzioni ottime globali** per il problema; nel caso in cui risulti invece $f(x) \leq f(y)$ soltanto per ogni y appartenente ad un *intorno* di x, allora la soluzione x si dice soluzione ottima **locale**.

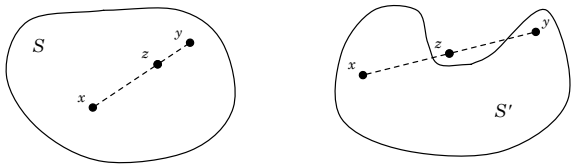

Figura 1: Un insieme convesso S ed un insieme S' non convesso (concavo): S' è tale che, fissata una coppia di punti $x, y \in S'$, esistono altri punti $z = \lambda x + (1-\lambda)y$ che, per alcuni valori di $\lambda \in [0,1]$, non appartengono ad S'

L'insieme A delle *soluzioni ammissibili* è l'insieme entro il quale devono essere cercate le soluzioni che rendono minima o massima la funzione obiettivo: la definizione di A è una caratteristica specifica di ciascun problema e consente di escludere punti dello spazio su cui sono definite le variabili del problema su cui non è utile o non ha senso cercare la soluzione. Non si deve necessariamente pensare all'insieme A come ad un sottoinsieme o un intervallo della retta reale: spesso infatti l'insieme A è un sottoinsieme di $\mathbb{R}^n$ o un insieme discreto definito in modo più articolato, come ad esempio un sottografo con determinate caratteristiche.

Ad esempio il problema della ricerca del cammino più breve tra due vertici u e v di un grafo G, può essere formulato come problema di ottimizzazione definendo l'insieme A delle soluzioni ammissibili come la collezione di tutti i cammini semplici in G dal vertice u al vertice v; la funzione obiettivo $f : A \to \mathbb{R}$ viene definita come la funzione che associa ad ogni cammino $p \in A$ la sua lunghezza. I cammini ottimi sono quelli che rendono minima la funzione obiettivo, ossia quei cammini da u a v con la lunghezza minima.

Nella formulazione di un problema di ottimizzazione un aspetto assai critico e rilevante è costituito dalla corretta definizione dell'insieme delle soluzioni ammissibili; è necessario circoscrivere questo sottoinsieme dello "spazio delle soluzioni", senza escludere punti dello spazio che possano corrispondere ad una soluzione ammissibile e dunque potenzialmente ottima e al tempo stesso cercando di ridurre al massimo l'insieme entro cui eseguire la ricerca delle soluzioni, per rendere più efficiente e meno oneroso in termini computazionali, il procedimento di calcolo adottato per individuare le soluzioni ottime.

In generale quindi l'insieme A delle soluzioni ammissibili viene definito attraverso un insieme di **vincoli** che limitano e circoscrivono l'insieme entro cui possono assumere valori le variabili del problema.

In particolare tra i problemi di ottimizzazione rivestono notevole importanza i **problemi di programmazione matematica**. In questo genere di problema i vincoli che circoscrivono l'insieme A delle soluzioni ammissibili sono costituiti da un certo numero, anche molto ampio, ma sempre finito, di equazioni e disequazioni; da qui il nome di questa classe di problemi. In generale quindi un problema di programmazione matematica viene definito come segue:

$$\begin{aligned} &\text{minimizzare } f(x) \quad \text{per } x \text{ tale che} \\ &g_i(x) \geq 0 \quad i = 1,2,\ldots,m \\ &h_j(x) = 0 \quad j = 1,2,\ldots,p \end{aligned} \tag{1}$$

Le funzioni g_i e h_j costituiscono i vincoli del problema.

Dati due punti dello spazio $x, y \in \mathbb{R}^n$ una **combinazione convessa** di x e y è un punto z dello spazio definito come $z = \lambda x + (1-\lambda)y$, con $\lambda \in \mathbb{R}$ e $0 \leq \lambda \leq 1$. Un insieme $S \subseteq \mathbb{R}^n$ è **convesso** se contiene qualsiasi combinazione convessa di ogni coppia di elementi distinti di S; in altri termini S è convesso

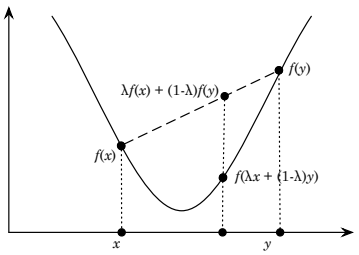

Figura 2: La funzione convessa calcolata nei punti di una combinazione convessa è minore o uguale alla combinazione convessa dei valori della funzione negli estremi

se per ogni $x, y \in S$, $x \neq y$, risulta $\lambda x + (1-\lambda)y \in S$ per ogni $0 \leq \lambda \leq 1$. In Figura 1 è rappresentato un insieme S convesso ed un insieme S' non convesso in $\mathbb{R}^2$.

Proposizione 1. *Se $S_1, S_2, \ldots, S_k$ sono insiemi convessi, allora anche $\cap_{i=1}^k S_i$ è un insieme convesso.*

Dimostrazione. Se $x, y \in \cap_{i=1}^k S_i$ allora $x, y \in S_i$, per ogni $i = 1, \ldots, k$. Per cui, visto che per ipotesi ogni insieme S_i è convesso, ogni combinazione convessa di x e y appartiene ad S_i e dunque appartiene anche a $\cap_{i=1}^k S_i$. □

Sia $S \subseteq \mathbb{R}^n$ un insieme convesso. La funzione $f : S \to \mathbb{R}$ è **convessa** in S se, per ogni $x, y \in S$, risulta

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y) \tag{2}$$

Se vale la disuguaglianza opposta, la funzione si dice **concava**. Evidentemente se $f(x)$ è convessa, allora $-f(x)$ è concava. In Figura 2 è rappresentato il grafico di una funzione convessa f definita a valori su $\mathbb{R}$. Un esempio di funzione convessa su $\mathbb{R}^2$ è dato dal paraboloide $f(x_1, x_2) = x_1^2 + x_2^2$.

Proposizione 2. *Sia $f(x)$ una funzione convessa sull'insieme convesso S. Allora, per ogni $t \in \mathbb{R}$, l'insieme $S_t = \{x \in S : f(x) \leq t\}$ è convesso.*

Dimostrazione. Dobbiamo mostrare che ogni combinazione convessa di elementi di S_t è ancora in S_t, ossia che la funzione $f(x)$, calcolata in ogni combinazione convessa di elementi di S_t, è minore o uguale a t. Siano $x, y \in S_t$ per qualche $t \in \mathbb{R}$. Per ipotesi (S è convesso) ogni combinazione convessa di x e y, $\lambda x + (1-\lambda)y$ è in S. Inoltre, siccome f è una funzione convessa, risulta

$$\begin{aligned} f(\lambda x + (1-\lambda)y) &\leq \lambda f(x) + (1-\lambda) f(y) \\ &\leq \lambda t + (1-\lambda) t \\ &= t \end{aligned}$$

Quindi effettivamente $f(\lambda x + (1-\lambda)y) \leq t$ e dunque $\lambda x + (1-\lambda)y \in S_t$. In conclusione, allora, S_t è un insieme convesso. □

Un **problema di programmazione convessa** è un problema di programmazione matematica espresso nella forma (1), in cui la funzione obiettivo $f(x)$ sia convessa, le funzioni $g_i(x)$ siano concave e le $h_j(x)$ siano lineari. Perché è importante distinguere tra i generici problemi di programmazione matematica quelli che sono dotati di una funzione obiettivo convessa? La risposta è semplice: se la funzione obiettivo è convessa allora ogni soluzione ottima locale è anche una soluzione ottima globale.

Proposizione 3. *In un problema di programmazione convessa ogni soluzione ottima locale è anche una soluzione ottima globale.*

Dimostrazione. Supponiamo che per un determinato problema di programmazione convessa (A, f) risulti $f(x) \leq f(y)$ per ogni $y \in N(x) \subseteq A$ (indichiamo con $N(x)$ un intorno del punto x) e che esista un punto $z \notin N(x)$, $z \in A$, tale che $f(x) \not\leq f(z)$. In questo caso x sarebbe una soluzione ottima locale, ma non globale. Dunque se così fosse, allora esisterebbe sicuramente un punto $y \in N(x)$ tale che possa essere espresso come combinazione convessa di x e z: $y = \lambda x + (1-\lambda)z$. In questo caso si avrebbe $f(z) < f(x) \leq f(y)$ contraddicendo così la convessità di f. □

3 Programmazione lineare

Le funzioni lineari sono un caso molto particolare di funzioni convesse. Se la funzione obiettivo ed i vincoli del problema di ottimizzazioni sono funzioni lineari, allora il problema di programmazione matematica che si ottiene è un **problema di programmazione lineare** (LP). È facile osservare che se la funzione obiettivo è lineare, allora i suoi punti di minimo o di massimo si trovano sicuramente sulla frontiera dell'insieme A delle soluzioni ammissibili.

Proprio su questo principio si basa il celebre metodo per la risoluzione di problemi di programmazione lineare proposto dal matematico americano George Dantzig nel 1947 e noto come *algoritmo del simplesso*. L'algoritmo proposto da Dantzig opera su un *simplesso*, ossia un politopo in n dimensioni con $n+1$ vertici (ad esempio, in uno spazio di dimensione $n = 3$, un simplesso è un tetraedro, il poliedro con il minor numero di vertici). I vincoli del problema di ottimizzazione definiscono la regione ammissibile, cioè l'insieme dei punti che soddisfano tutti i vincoli del problema. Nel caso della programmazione lineare la regione ammissibile è un politopo (un poligono nel piano, un poliedro nello spazio), che può essere vuoto (nel caso in cui non esistono soluzioni al problema), limitato o illimitato. La funzione obiettivo che deve essere minimizzata o massimizzata esprime il "costo" di ogni soluzione tenendo conto dei vincoli (ossia calcolando la soluzione all'interno del poliedro). L'algoritmo del simplesso è in grado di determinare di che tipo di poliedro si tratta e di individuare la soluzione ottima, che è, sotto opportune ipotesi, un vertice del poliedro, nel caso in cui il problema abbia una soluzione ottimale finita.

Tra i problemi di programmazione lineare è opportuno distinguere i **problemi di programmazione lineare intera** (PLI): in questo caso alcune delle variabili del problema sono vincolate ad assumere soltanto valori interi. Questo vincolo ulteriore riduce drasticamente il numero di punti presenti nella regione ammissibile.

Il problema di programmazione lineare diventa un problema di **ottimizzazione combinatoria** se *tutte* le variabili sono vincolate ad assumere soltanto valori interi.

Quando in un problema di programmazione lineare intera si eliminano i "vincoli di integrità", ossia i vincoli che impongono $x \in \mathbb{Z}^n$, si ottiene un problema di programmazione lineare detto **rilassamento continuo**. Se, inoltre, accade che le soluzioni ottime del problema originario sono anche soluzioni ottime del suo rilassamento continuo, si dice che il problema gode della *proprietà di integralità*. In effetti la proprietà di integralità riguarda l'insieme ammissibile del problema più che il problema in sè. Infatti la funzione obiettivo del rilassamento continuo ha lo stesso valore di quella del problema originario sul suo insieme ammissibile e, inoltre, poiché è un problema di programmazione lineare, ha sempre una soluzione ottima di vertice.

Per chiarire meglio questi concetti, vediamo di seguito alcuni problemi classici di ottimizzazione combinatoria che possono essere formulati come problemi di Programmazione Lineare e di Programmazione Lineare Intera.

3.1 Problema della bisaccia

Il *problema della bisaccia binario* (*knapsack problem*, in inglese) può essere posto come segue: abbiamo a disposizione un sacco, una bisaccia, in grado di sopportare un carico pari a P, con cui siamo intenzionati a trasportare un certo numero di oggetti di peso e di valore diverso, selezionandoli da una collezione di n elementi, in modo tale da massimizzare il valore complessivo degli oggetti selezionati, ma senza rompere la bisaccia riempiendola con un peso eccessivo, superiore a P. Naturalmente ogni oggetto ha un peso intero ben definito e non è possibile rompere gli oggetti in modo da costituire oggetti più piccoli di peso inferiore: ciascun oggetto può essere preso per intero oppure deve essere lasciato al suo posto, rinunciando a trasportarlo. Qual è il modo migliore con cui è possibile operare la scelta degli oggetti da inserire nella bisaccia? Il problema può essere formulato come un problema di programmazione lineare intera indicando con c_i e p_i rispettivamente il valore e il peso dell'i-esimo oggetto. Con la variabile binaria $x_i \in \{0,1\}$ indicheremo se l'oggetto i-esimo è stato scelto ($x_i = 1$) oppure se è stato scartato ($x_i = 0$). In questo modo la soluzione ottima del problema è una sequenza $x = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n$ con cui possiamo identificare il contenuto della bisaccia; il problema di ottimizzazione può essere posto nei seguenti termini:

$$\begin{aligned} \text{massimizzare } f(x) &= \sum_{i=1}^{n} c_i x_i \\ \sum_{i=1}^{n} p_i x_i &\leq P \\ x_i &\in \{0,1\} \quad i = 1,2,\ldots,n \end{aligned} \tag{3}$$

Le variabili del problema, $x_1, \ldots, x_n$, non soltanto sono intere, ma addirittura sono vincolate ad assumere valori binari: $x_i = 0$ oppure $x_i = 1$. In questi casi si parla di problema di programmazione lineare intera "0-1". Il problema è chiaramente di tipo combinatorio: le soluzioni ammissibili sono in numero finito e possono essere anche prodotte in modo esaustivo, costruendo tutte le sequenze binarie con n caratteri; tuttavia questo approccio conduce ad un algoritmo di complessità esponenziale nella dimensione n dell'istanza del problema, dal momento che il numero di stringhe binarie con n caratteri (o, come abbiamo visto in precedenza, l'insieme delle parti di un insieme di cardinalità n) sono 2^n.

3.2 Problema dell'assegnazione di task

Un problema classico di pianificazione riguarda l'assegnazione di *task* ai processori di un calcolatore parallelo, alle macchine di una rete di calcolatori che cooperano per la soluzione di uno stesso problema o ad un gruppo di persone. Supponiamo che n processori debbano svolgere m *task*. Ciascuno di essi deve essere svolto esattamente da un processore ed inoltre, ciascun processore può svolgere al al massimo un task. Indichiamo con c_{ij} il costo dell'utilizzo del processore i per l'esecuzione del task j. Il problema chiede di assegnare i task alle varie risorse (processori, calcolatori o persone, a seconda del contesto applicativo) minimizzando il costo totale per la realizzazione di tutti gli m task previsti. Per formulare questo problema in termini di programmazione lineare intera, introduciamo le variabili binarie x_{ij}, per $i = 1,\ldots,n$ e $j = 1,\ldots,m$ corrispondenti all'evento "ij" (esecuzione del task j da parte della risorsa i) definite come segue $x_{ij} = 1$ se il task j viene eseguito dalla risorsa i e $x_{ij} = 0$ altrimenti.

Poiché esattamente una ed una sola risorsa deve essere assegnata al task j, avremo i seguenti vincoli:

$$\sum_{i=1}^{n} x_{ij} = 1, \text{ per ogni } j = 1,2,\ldots,m$$

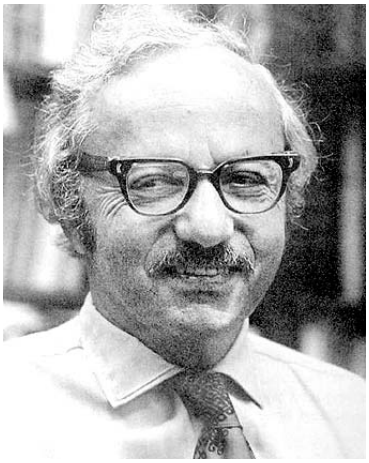

Figura 3: George Bernard Dantzig (1914–2005)

Al tempo stesso, visto che per ipotesi ogni risorsa non può svolgere più di un task, si avranno anche i seguenti vincoli:

$$\sum_{j=1}^{m} x_{ij} = 1, \text{ per ogni } i = 1,2,\ldots,n$$

È facile verificare che un vettore $x \in \{0,1\}$ che soddisfa tutti i vincoli appena descritti individua un assegnamento ammissibile di risorse ai task. La funzione obiettivo, che ovviamente deve essere minimizzata, è la seguente:

$$\sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij}$$

4 Il metodo del simplesso

Il *metodo del simplesso* è un algoritmo generale per la risoluzione di problemi di programmazione lineare proposto nel 1947 dal matematico americano George Dantzig.[1] La sua rilevanza, oltre che storica, anche dal punto di vista applicativo, è fondata proprio sulla generalità dell'approccio, piuttosto che sulla sua efficienza: l'algoritmo del simplesso permette di risolvere qualsiasi problema di ottimizzazione formulato come problema di programmazione lineare, anche se computazionalmente è piuttosto pesante.

Possiamo riformulare un problema di programmazione lineare partendo dall'espressione (1), formalizzandolo come segue: data una matrice $A \in \mathbb{R}^{m\times n}$ e i vettori colonna $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{c} \in \mathbb{R}^n$, si chiede di trovare un vettore colonna $\mathbf{x} \in \mathbb{R}^n$ tale che $A\mathbf{x} \leq \mathbf{b}$ e $\mathbf{c}^t\mathbf{x}$ sia minimo (problema in forma di minimizzazione). Naturalmente il vettore $\mathbf{c}$ rappresenta il "costo" associato a ciascuna variabile x_i della soluzione del problema (o il "profitto", nel caso di problemi in forma di massimizzazione) e la funzione obiettivo è espressa quindi come $f(\mathbf{x}) = \mathbf{c}^t\mathbf{x}$. La matrice A è la matrice dei coefficienti delle disequazioni lineari con cui viene definito l'insieme delle soluzioni ammissibili del problema. L'espressione (1)

[1] Per una trattazione più completa dell'argomento, che in queste pagine viene solo tratteggiato nelle sue linee più generali, si rimanda ai testi citati tra i riferimenti bibliografici ed in particolare [3], [5] e [6].

può così essere riformulata come segue:

$$\begin{aligned} \text{minimizzare } f(\mathbf{x}) = \mathbf{c}^{\mathrm{t}}\,\mathbf{x} \quad & \text{per } \mathbf{x} \text{ tale che} \\ A\mathbf{x} &\leq \mathbf{b} \\ A \in \mathbb{R}^{m\times n}, \mathbf{b} \in \mathbb{R}^m, \mathbf{c} &\in \mathbb{R}^n \end{aligned} \tag{4}$$

L'insieme di disequazioni lineari $A\mathbf{x} \leq \mathbf{b}$ definisce un poliedro P come intersezione dei semispazi definiti da ciascuna disequazione. È all'interno di P che sono da ricercare le soluzioni ottime del problema, tra tutte le soluzioni ammissibili.

Esistono due casi degeneri per un problema di programmazione lineare: il caso in cui il problema non ammette nessuna soluzione, quando cioè i vincoli producono un poliedro nullo, $P = \emptyset$, o il caso in cui il problema ammette infinite soluzioni, quando il poliedro P è illimitato. Quando il problema di programmazione lineare è ben posto e dunque P non è né nullo, né illimitato, diremo che P è un *politopo*. Un *simplesso*, il termine che dà il nome all'algoritmo di Dantzig, è un politopo di $n+1$ vertici in n dimensioni: un segmento in dimensione 1, un triangolo in dimensione 2 e un tetraedro in dimensione 3.

Cerchiamo di formalizzare meglio questi concetti. Innanzi tutto chiamiamo *vertice* di un poliedro P un punto $\mathbf{x} \in P$ tale che non esistono altri due punti $\mathbf{x}', \mathbf{x}'' \in P$, con $\mathbf{x} \neq \mathbf{x}', \mathbf{x}''$ e $\mathbf{x} \in [\mathbf{x}', \mathbf{x}'']$.

Non tutti i poliedri ottenuti come intersezioni di semispazi definiti mediante disequazioni lineari hanno almeno un vertice. Ad esempio se consideriamo un unico semispazio di $\mathbb{R}^2$, questo non ha vertici. Dunque il poliedro $P = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \geq \mathbf{b}\}$ non ha vertici se la matrice A ha un numero di righe minore di n: in tal caso non è possibile trovare n vincoli linearmente indipendenti che circoscrivano un poliedro chiuso.

Un poliedro P non ha vertici se contiene rette: diremo che un poliedro P contiene una retta se esiste un punto $\tilde{\mathbf{x}} \in P$ e un vettore non nullo $\mathbf{d} \in \mathbb{R}^n$ tale che $\tilde{\mathbf{x}} + \lambda\mathbf{d} \in P$ per ogni $\lambda \in \mathbb{R}$.

Possiamo infine enunciare il seguente *Teorema fondamentale della Programmazione Lineare*:

Teorema 1. *Si consideri il problema di Programmazione Lineare*

$$\begin{aligned} \min \mathbf{c}^t\,\mathbf{x} \\ A\mathbf{x} = \mathbf{b} \end{aligned}$$

Supponiamo che il poliedro $P = \{\mathbf{x} \in \mathbb{R}^n$ tale che $A\mathbf{x} = \mathbf{b}\}$ non contenga rette. Allora una e una sola delle seguenti tre affermazioni è vera:

1. *il problema è inammissibile, ovvero il poliedro P è vuoto;*
2. *il problema è illimitato inferiormente;*
3. *il problema ammette soluzioni ottime e almeno una di queste è un vertice del poliedro P.*

L'algoritmo del simplesso non consiste altro che in una "visita intelligente" dei vertici di P; il teorema fondamentale della programmazione lineare garantisce infatti che la soluzione ottimale del problema, se esiste, è situata in uno di questi vertici. Muovendosi sugli spigoli del poliedro, da un vertice ad un altro, il metodo del simplesso cerca di individuare una soluzione ottima, che massimizzi o minimizzi la funzione obiettivo.

Consideriamo la matrice $A \in \mathbb{R}^{m\times n}$; se I è un insieme di indici di riga di A, indichiamo con A_I la sottomarine di A composta con le sole righe in I; analogamente se $\mathbf{b}$ è un vettore, indichiamo con $\mathbf{b}_I$ il sottovettore composto dalle sole componenti di indice in I. Indichiamo con $\mathbf{a}_i$ la riga $A_{\{i\}}$ e con

Algoritmo 1 SIMPLESSO(A, $\mathbf{b}$, $\mathbf{c}$)

Input: Una matrice di coefficienti $A \in \mathbb{R}^{m\times n}$, il vettore dei termini noti $\mathbf{b} \in \mathbb{R}^m$, il vettore dei costi $\mathbf{c}^{\mathrm{t}} \in \mathbb{R}^n$, un vertice $\mathbf{x} \in P = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq \mathbf{b}\}$

Output: Un vertice $\mathbf{x}^* \in P$ tale che $\mathbf{c}^{\mathrm{t}}\,\mathbf{x}^* = \max_{\mathbf{x}\in P}\{\mathbf{c}^{\mathrm{t}}\,\mathbf{x}\}$, oppure un vettore $\mathbf{w} \in \mathbb{R}^n$ con $A\mathbf{w} \leq 0$ e $\mathbf{c}^{\mathrm{t}}\,\mathbf{w} > 0$ (il poliedro P è illimitato)

1: Scegli un insieme di n indici di riga I tali che A_I è non singolare e $A_I\mathbf{x} = \mathbf{b}_I$
2: Calcola $\mathbf{c}\ (A_I)^{-1}$ e aggiungi gli zeri necessari per ottenere un vettore $\mathbf{y}$ tale che $\mathbf{c} = \mathbf{y}A$ e tutti gli elementi di $\mathbf{y}$ fuori da I siano nulli
3: **se** $\mathbf{y} \geq 0$ **allora**
4: **return** $\mathbf{x}$ e $\mathbf{y}$; stop
5: **fine-condizione**
6: Sia i il minimo indice per cui risulta $y_i < 0$
7: Sia $\mathbf{w}$ la colonna di $-(A_I)^{-1}$ con indice i, tale che $A_{I\setminus\{i\}}\mathbf{w} = 0$ e $\mathbf{a}_i\mathbf{w} = -1$
8: **se** $A\mathbf{w} \leq 0$ **allora**
9: **return** $\mathbf{w}$; stop
10: **fine-condizione**
11: Sia $\lambda := \min_{j=1,\ldots,m}\left\{\frac{b_j - \mathbf{a}_j\mathbf{x}}{\mathbf{a}_j\mathbf{w}} : \mathbf{a}_j\mathbf{w} > 0\right\}$ e sia j il più piccolo indice di riga con cui si ottiene questo minimo
12: Sia $I := (I \setminus \{i\}) \cup \{j\}$ e $\mathbf{x} := \mathbf{x} + \lambda\mathbf{w}$
13: Vai al passo 2

$b_i = \mathbf{b}_{\{i\}}$. Utilizzando queste notazioni formalizziamo il Metodo del Simplesso con lo pseudo codice dell'Algoritmo 1.

È possibile dimostrare che l'Algoritmo del Simplesso termina al massimo dopo $\binom{m}{n}$ iterazioni del ciclo 2–13. Se l'algoritmo restituisce $\mathbf{x}$ e $\mathbf{y}$ al passo 4, allora $\mathbf{x}$ e $\mathbf{y}$ sono i vettori che rappresentano le soluzioni ottime per i seguenti problemi di programmazione lineare, con $\mathbf{c}\,\mathbf{x} = \mathbf{y}\,\mathbf{b}$:

$$\max\{\mathbf{c}\,\mathbf{x} : A\mathbf{x} \leq \mathbf{b}\} \tag{5}$$

$$\min\{\mathbf{y}\,\mathbf{b} : \mathbf{y}^{\mathrm{t}}A = \mathbf{c}^{\mathrm{t}}, \mathbf{y} \geq 0\} \tag{6}$$

Infine, se l'algoritmo restituisce il vettore $\mathbf{w}$ al passo 9, allora $\mathbf{c}\mathbf{w} > 0$ e il problema di programmazione lineare (5) è illimitato.

Il problema di programmazione lineare espresso da (5) viene chiamato problema *primale*, mentre il problema di programmazione lineare (6), associato al precedente, viene chiamato problema *duale*.

5 Programmazione lineare per problemi di ottimizzazione su grafi

Per concludere questa breve panoramica sui problemi di programmazione matematica e su quelli di programmazione lineare in particolare, vediamo alcune applicazioni della programmazione lineare alla teoria dei grafi. Una volta impostato il problema in termini di programmazione lineare è possibile calcolarne una soluzione con l'algoritmo del simplesso, che garantisce l'identificazione di una soluzione ottima, al costo di una complessità computazionale più elevata di un algoritmo specificamente progettato per risolvere un determinato problema (es.: il problema del cammino minimo tra due vertici di un grafo). Di seguito consideriamo alcuni problemi classici di ottimizzazione combinatoria con l'obiettivo di offrirne una formulazione in termini di programmazione lineare intera:

1. minimo ricoprimento di vertici per gli spigoli di un grafo G;
2. massima clique di un grafo G;
3. cammino di costo minimo dal vertice s al vertice t su un grafo G;
4. albero ricoprente di costo minimo di un grafo G.

I problemi saranno formulati come problemi di programmazione lineare intera "$0-1$", ossia con variabili x_i che assumono valori nell'insieme $\{0,1\}$; infatti, nel caso di problemi di ottimizzazione combinatoria, la soluzione è data sempre da una "scelta" degli elementi che costituiscono l'istanza del problema: una scelta di alcuni vertici di un grafo, una scelta di alcuni degli spigoli di un grafo, in generale una scelta di alcuni elementi di un insieme discreto, per cui è comodo rappresentare con la variabile x_i il fatto di aver scelto ($x_i = 1$) oppure di aver scartato ($x_i = 0$) l'elemento i-esimo dell'insieme.

5.1 Minimo insieme di copertura dei vertici

Dato un grafo $G = (V,E)$ un *ricoprimento di vertici* è un sottoinsieme $C \subseteq V$ tale che ogni spigolo di G sia incidente almeno a un vertice di C. Il problema di ottimizzazione combinatoria che si intende risolvere chiede di calcolare un insieme C di copertura dei vertici di G, di cardinalità minima.

Per modellizzare il problema introduciamo le seguenti variabili:

$$\forall\, v_i \in V,\ x_i = \begin{cases} 1 \text{ se } v_i \text{ è nel ricoprimento } C \\ 0 \text{ altrimenti} \end{cases}$$

Si tratta dunque di minimizzare il numero di vertici nel ricoprimento, ovvero la funzione: $\sum_{v_i \in V} x_i$. La condizione che garantisce che C sia un ricoprimento è che per ogni spigolo $(v_i, v_j) \in E(G)$ almeno uno tra v_i e v_j sia nel ricoprimento C. Formalizzando si ha dunque il seguente vincolo: $\forall (v_i, v_j) \in E(G),\ x_i + x_j \geq 1$.

Riassumendo, il problema del minimo ricoprimento può essere formalizzato nel modo seguente come problema di programmazione lineare intera:

$$\begin{cases} \min \sum_{v_i \in V} x_i \\ \forall\, (v_i, v_j) \in E,\ x_i + x_j \geq 1 \\ \forall\, v_i \in V\ x_i \in \{0,1\} \end{cases}$$

5.2 Clique massima

Dato un grafo $G = (V,E)$ una *clique* di G è un sottoinsieme $C \subseteq V$ tale che ogni coppia di vertici in C sia collegata da uno spigolo in G (C induce un sottografo completo su G). Dato un grafo G si vuole quindi individuare una clique di dimensione massima su G.

Per modellizzare il problema in termini di programmazione lineare introduciamo le seguenti variabili:

$$\forall\, v_i \in V,\ x_i = \begin{cases} 1 \text{ se } v_i \text{ è nella clique} \\ 0 \text{ altrimenti} \end{cases}$$

Si tratta dunque di massimizzare il numero di vertici nella clique, ovvero la funzione: $\sum_{v_i \in V} x_i$. La condizione che garantisce che C sia una clique è che per ogni coppia (v_i, v_j) che non sia uno spigolo di G almeno uno tra v_i e v_j non sia in C. In termini formali si ha dunque il seguente vincolo: $\forall\, (v_i, v_j) \notin E(G),\ x_i + x_j \leq 1$.

Riassumendo, il problema della massima clique può essere formulato come problema di programmazione lineare intera nel modo seguente:

$$\begin{cases} \max \sum_{v_i \in V} x_i \\ \forall\ (v_i, v_j) \notin E,\ x_i + x_j \leq 1 \\ \forall\ v_i \in V,\ x_i \in \{0,1\} \end{cases}$$

5.3 Cammino di costo minimo

Sia $G = (V,E)$ un grafo con dei pesi non negativi assegnati agli spigoli: per ogni $(v_i, v_j) \in E(G)$ $d_{ij} \geq 0$ sia il peso corrispondente. Dati due vertici $s,t \in V$, si vuole trovare un cammino p da s a t per cui la somma dei pesi assegnati agli spigoli sia minima.

Per formulare il problema in termini di programmazione lineare, si introducono le seguenti variabili, questa volta una per ogni spigolo:

$$\forall\ (v_i, v_j) \in E(G)\ x_{ij} = \begin{cases} 1 \text{ se } (v_i, v_j) \text{ è nel cammino } p : s \rightsquigarrow t \\ 0 \text{ altrimenti} \end{cases}$$

Per trovare una soluzione del problema dobbiamo quindi minimizzare la seguente funzione obiettivo: $\sum_{(v_i,v_j) \in E} d_{ij} x_{ij}$.

Affinché un insieme di spigoli del grafo G costituisca un cammino da s a t bisogna assicurarsi che esattamente uno spigolo del cammino esca da s e nessuno entri in esso, esattamente uno spigolo del cammino entri in t e nessuno esca da esso e che per tutti gli altri vertici il numero di spigoli del cammino uscenti sia pari a quello di spigoli entranti. Formalizzando si ottengono i seguenti vincoli:

$$\sum_{(v_i,v_h) \in E} x_{hj} - \sum_{(v_j,v_i) \in E} x_{ij} = \begin{cases} -1, & v_h = s \\ 1, & v_h = t \\ 0, & \forall v_h \in V \setminus \{s,t\} \end{cases}$$

Riassumendo, il problema del cammino minimo può essere formalizzato nel modo seguente come un problema di programmazione lineare intera:

$$\begin{cases} \sum_{(v_i,v_j) \in E} d_{ij} x_{ij} \\ \sum_{(v_i,v_h) \in E} x_{hj} - \sum_{(v_j,v_i) \in E} x_{ij} = -1, \text{ se } v_h = s \\ \sum_{(v_i,v_h) \in E} x_{hj} - \sum_{(v_j,v_i) \in E} x_{ij} = 1, \text{ se } v_h = t \\ \sum_{(v_i,v_h) \in E} x_{hj} - \sum_{(v_j,v_i) \in E} x_{ij} = 0, \text{ se } v_h \in V \setminus \{s,t\} \\ \forall\ (v_i, v_j) \in E,\ x_{ij} \in \{0,1\} \end{cases}$$

5.4 Albero ricoprente di costo minimo

Un albero ricoprente (*spanning tree*) di un grafo $G = (V,E)$ è un albero $T = (V,E')$, con $E' \subseteq E$, ossia un albero con gli stessi vertici di G e $n-1$ spigoli di G, affinché T sia un grafo connesso e aciclico (un albero). Se associamo un peso non negativo $w(u,v)$ ad ogni spigolo $(u,v) \in E(G)$, è possibile definire il problema di ottimizzazione combinatoria che chiede di costruire l'albero ricoprente di G di peso minimo, ossia l'albero T^* tale che

$$\sum_{(u,v) \in E(T^*)} w(u,v) = \min_{T \text{ che ricopre } G} \sum_{(u,v) \in E(T)} w(u,v)$$

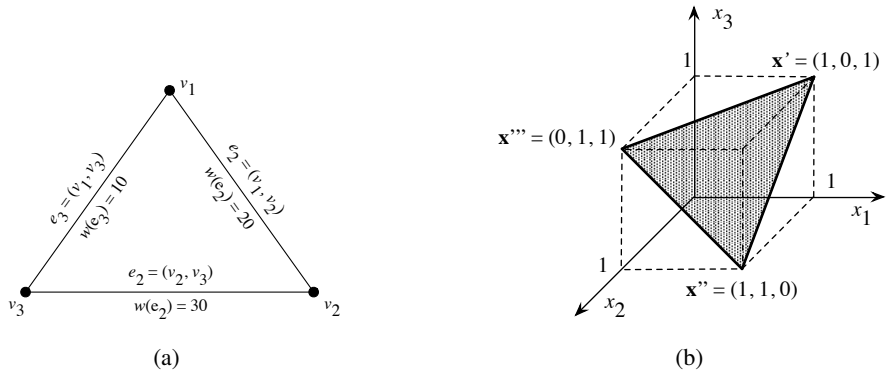

Figura 4: Un grafo pesato $G = (V,E)$ e il politopo con la regione delle soluzioni ammissibili per il problema MST.

Anche in questo caso, per formulare il problema in termini di programmazione lineare, si introducono le seguenti variabili associate agli spigoli del grafo:

$$\forall\, e_i \in E(G)\; x_i = \begin{cases} 1 \text{ se } e_i \in E(T) \\ 0 \text{ altrimenti} \end{cases}$$

In questo modo possiamo formulare il problema del minimo albero ricoprente (MST – *minimum spanning tree*) come segue:

$$\begin{aligned} \min \quad & f(\mathbf{x}) = \textstyle\sum_{i=1}^{m} w(e_i)x_i \\ \text{per } \mathbf{x} \text{ tale che} \quad & x_1 + x_2 + \cdots + x_m = n-1 \\ & x_i \geq 0 \;\text{ per } i = 1,2,\ldots,m \\ & x_i \leq 1 \;\text{ per } i = 1,2,\ldots,m \end{aligned}$$

Per aiutare a visualizzare il significato geometrico di questa formulazione, consideriamo il grafo $G = (V,E)$ definito ponendo $V = \{v_1, v_2, v_3\}$, $E = \{e_1 = (v_1, v_2), e_2 = (v_2, v_3), e_3 = (v_1, v_3)\}$; supponiamo anche che $w(e_1) = 20$, $w(e_2) = 30$ e $w(e_3) = 10$. Il grafo è rappresentato in Figura 4(a).

Effettuando un rilassamento continuo del problema, possiamo rappresentare l'insieme delle soluzioni ammissibili con il poliedro (triangolo) evidenziato in Figura 4(b), ossia il poliedro a 2 dimensioni collocato nello spazio a 3 dimensioni, con vertici nei punti $\mathbf{x}' = (1,0,1)$, $\mathbf{x}'' = (1,1,0)$ e $\mathbf{x}''' = (0,1,1)$. Per il vincolo di integrità delle variabili del problema (dobbiamo scegliere uno spigolo o non sceglierlo per nulla, non possiamo prenderne solo un pezzo!), le soluzioni del problema si trovano proprio nei vertici del politopo costituito dal triangolo. In particolare, come si può osservare facilmente senza la necessità di applicare l'Algoritmo del Simplesso, la soluzione ottima è costituita dal punto $\mathbf{x}'' = (1,0,1)$: questo punto rappresenta l'albero ricoprente costruito scegliendo gli spigoli e_1 ed e_3, con peso complessivo $f(\mathbf{x}'') = w(e_1) + w(e_3) = 20 + 10 = 30$.

Riferimenti bibliografici

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Introduzione agli algoritmi e strutture dati*, seconda edizione, McGraw-Hill, 2005.

[2] Ottavio D'Antona, *Introduzione alla Matematica Discreta*, Apogeo, 1999.

[3] Bernhard Korte, Jens Vygen, *Ottimizzazione Combinatoria*, Springer, 2011.

[4] Marco Liverani, *Qual è il problema? Metodi, strategie risolutive, algoritmi*, Mimesis, 2005.

[5] Christos H. Papadimitriou, Kenneth Steiglitz, *Combinatorial Optimization – Algorithms and Complexity*, Dover Publications Inc., 1998.

[6] Antonio Sassano, *Modelli e algoritmi della Ricerca Operativa*, Franco Angeli, 1999.

[7] Paolo Serafini, *Ottimizzazione*, Zanichelli, 2000.