Proposta Tesi, tirocinio Argo Software

Angelo Battaglia

April 25, 2021

Politecnico
di Torino

1 Introduzione

Citando Luenberger and Ye [1], un problema di ottimizzazione riguarda l'allocazione delle risorse e l'analisi della complessità di un problema. L'ottimizzazione si suddivide in tre sottoinsiemi: Linear Programming, Unconstrained Problems e Constrained Problems. In seguito, utilizzerò questi nomi sia in Italiano che in Inglese. Il successo delle tecniche di ottimizzazione della programmazione lineare risiedono soprattutto nella formulazione del problema, piuttosto che nel metodo utilizzato per risolverlo. Un problema con un grande numero di vincoli, infatti, presenta usualmente delle caratteristiche di linerità.

1.1 Linear Programming

Siano date i entità, alle quali associamo

$$w_i$$

pesi. Un problema base che riguarda la programmazione lineare è:

$$massimizzare \quad w_1x_1 + w_2x_2$$

$$soggetto\ al\ vincolo \quad x_1 + x_2 \leq B$$

$$con \quad x_1 \geq 0,\ x_2 \geq 0$$

1.2 Unconstrained Problems

Problemi senza vincoli, ovvero unconstrained problems, sono quelli che considerano tutto il fascio di decisioni possibili. L'assenza di vincoli amplia la portata del problema, lo generalizza, aumentandone i gradi di libertà. Inoltre, aggiungere dei vincoli a questa classe di problemi è un'operazione che non presenza, di solito, particolari criticità. A livello teorico, i problemi senza vincoli assumono un particolare valore teorico, che successivamente va a concretizzarsi nella quotidianità dei problemi con vincoli.

1.3 Constrained Problems

I problemi ai vincoli, detti Constrained Problems, trovano più utilità nelle applicazioni. I vincoli vengono imposti nei problemi complessi per semplificarli, e per discretizzarli. Ogni vincolo riduce la portata del problema. Un problema generale che riguarda la programmazione matematica è:

$$minimizzare \quad f(\mathbf{x})$$

$$soggetto\ ai\ vincoli \quad h_i(\mathbf{x}) = 0,\ i = 1, 2, ...,\ m$$

$$g_j(\mathbf{x}) = 0,\quad j = 1, 2, ...,\ p$$

$$con\ x \in S$$

In questa definizione, $\mathbf{x}$ è un vettore n-dimensionale nelle incognite x_i, della forma $\mathbf{x} = (x_1, x_2, ..., x_n)$, ed

$$f\ ,\ h_i,\ i = 1, 2, ...,\ m,\ g_j,\ j = 1, 2, ...,\ p$$

sono delle funzioni a valori reali delle variabili $x_1,\ x_2,\ ...,\ x_n$. Una misura ovvia della complessità di un problema è la sua dimensione. La dimensione di un problema è misurata nei termini del numero di incognite e di vincoli. Come ci si può aspettare, i problemi trovano una più facile soluzione quando la potenza dei calcolatori aumenta e la teoria matematica viene raffinata. Un problema di larga scala viene ad avere migliaglia o milioni di variabili e vincoli. Un possibile strada da seguire, in termini di software, è *Gecode, Generic Contraint Development Environment* [2].

1.4 Algoritmi Iterativi e convergenza

La caratteristica più importante di un super-computer è la sua abilità di computare operazioni ripetitive in maniera efficiente. Per questa ragione, molti molti algoritmi di ottimizzazione, per loro natura, sono iterativi. Tipicamente si parte da un vettore iniziale x_0 per poi ottenerne uno, tramite un algoritmo, x_1. Il processo è ripetuto finché una soluzione migliore x_2 viene trovata. Continuando finché Una sequenza di punti sempre migliori è trovata $(x_0,\ x_1, ...,\ x_k, ...)$ che si avvicini alla soluzione x^*. Per i problemi di programazione lineare risolti dal metodo del simplesso, la sequenza generata è di lunghezza finita, e raggiunge la soluzione dopo un finito numero di passi, anche se non inizialmente specificato. Per la programmazione non lineare, la sequenza non raggiunge il punto x^* soluzione, ma converge verso esso, finché il processo termina ad un punto sufficientemente vicino al punto della soluzione.

The theory of iterative algorithms can be divided into three:

1) The first is concerned with the creation of the algorithms themselves. Algorithms are not conceived arbitrarily, but are based on a creative examination of the programming problem, its inherent structure, and the efficiencies of digital computers. 2) The second aspect is the verification that a given algorithm will in fact generate a sequence that converges to a solution point. This aspect is referred to as global convergence analysis, since it addresses the important question of whether the algorithm, when initiated far from the solution point, will eventually converge to it. 3) The third aspect is referred to as local convergence analysis or complexity analysis and is concerned with the rate at which the generated sequence of points converges to the solution. One cannot regard a problem as solved simply because an algorithm is known which will converge to the solution, since it may require an ex- orbitant amount of time to reduce the error to an acceptable tolerance. It is essential when prescribing algorithms that some estimate of the time required be available. It is the convergence-rate aspect of the theory that allows some quantitative evaluation and comparison of different algorithms, and at least crudely, assigns a measure of tractability to a problem. A modern-day technical version of Confucius' most famous saying, and one which represents an underlying philosophy of this book, might be, "One good the- ory is worth a thousand computer runs." Thus, the convergence properties of an it- erative algorithm can be estimated with confidence either by performing numerous computer experiments on different problems or by a simple well-directed theoreti- cal analysis. A simple theory, of course, provides invaluable insight as well as the desired estimate. For linear programming using the simplex method, solid theoretical statements on the speed of convergence were elusive, because the method actually converges to an exact solution in a finite number of steps. The question is how many steps might be required. This question was finally resolved when it was shown that it was possible for the number of steps to be exponential in the size of the program. The situation is different for interior point algorithms, which essentially treat the problem by introducing nonlinear terms, and which therefore do not generally obtain a solution in a finite number of steps but instead converge toward a solution.

For nonlinear programs, including interior point methods applied to linear programs, it is meaningful to consider the speed of convergence. There are many dif- ferent classes of nonlinear programming algorithms, each with its own convergence characteristics. However, in many cases the convergence properties can be deduced analytically by fairly simple means, and this analysis is substantiated by compu- tational experience. Presentation of convergence analysis, which seems to be the natural focal point of a theory directed at obtaining specific answers, is a unique feature of this book.

1.5 Convergence-rate theory

Abbiamo due aspetti della convergence-rate theory. Il primo è generalmente conosciuto come analisi della complessità, e concerne la velocità dell'algoritmo di convergenza, distinguendo tra tempo polinomiale e tempo non polinomiale. Il secondo aspetto provvede l'analisi di quanto il metodo velocemente converge nei suoi passi finali, e può provvedere un paragone dei vari metodi.

2 Metodo del Simplesso

3 Tabu Search

3.1 Simulated Annealing

Come cita l'articolo di Bertsimas and Tsitsiklis [3], il Simulated Annealing è un algoritmo di ottimizzazione che usa diversi strumenti matematici, come

4 Approccio Greedy

5 Algoritmi Genetici

6 Bin Packing

Bibliografia e crediti

[1] David G. Luenberger and Yinyu Ye. *Linear and Nonlinear Programming*. Springer, 2015.

[2] *Gecode, Generic Contraint Development Environment*. Gecode. 2021. URL: `https://www.gecode.org/index.html`.

[3] Dimitris Bertsimas and John Tsitsiklis. "Simulated Annealing". In: *Statistical Science* 8.5 (1993), pp. 10–15.