ALGORITMI EURISTICI

Luca Cappelletti
Prof. Roberto Cordone

6 CFU

2019
Informatica Magistrale
Università degli studi di Milano
Italia
19 marzo 2019

Indice

1 Introduzione

1.1 In cosa consiste l'esame

L'esame consiste in un orale ed un progetto software limitato temporalmente ad appello. L'orale inizia scegliendo un problema caratterizzato nel corso e viene fatto successivamente alla consegna del progetto.

1.2 Problemi tipici

I problemi vengono catalogati in base alla natura della loro soluzione.

Problema di decisione: la soluzione è vero o falso.

Problema di conteggio: la soluzione è il numero dei sottosistemi che soddisfano certe condizioni.

Problema di ottimizzazione: la soluzione è il valore minimo o massimo di una funzione obbiettivo definita su sottoinsiemi che soddisfano certe condizioni.

Problema di ricerca: la soluzione è la descrizione formale di un sottosistema che soddisfa certe condizioni.

Problema di enumerazione: la soluzione è la descrizione formale di tutti i sottosistemi che soddisfano certe condizioni.

1.3 Classificazione delle euristiche

Definizione 1.3.1 (Euristiche costruttive/distruttive). Sono un tipo di euristiche che partono da un sottoinsieme ovvio (l'insieme intero o vuoto) ed aggiungono o tolgono elementi sino ad ottenere la soluzione desiderata.

Definizione 1.3.2 (Euristiche di ricerca locale). Si tratta di un tipo di euristiche che partono da una soluzione ottenuta in qualsiasi modo e scambiano elementi fino a ottenere la soluzione desiderata.

Definizione 1.3.3 (Euristiche di ricombinazione). Si tratta di un insieme di euristiche che partono da una popolazione di soluzioni ottenuta in qualsiasi modo e ricombinano soluzioni diverse producendo una nuova popolazione.

Definizione 1.3.4 (Euristiche a convergenza). Si tratta di un insieme di euristiche che associano a ogni elemento del set un valore frazionario tra 0 e 1 e lo aggiornano iterativamente finché converge a $\{0,1\}$.

1.4 Rischi tipici

Atteggiamento referenziale o modaiolo: farsi dettare l'approccio dal contesto sociale.

Atteggiamento magico: credere all'efficacia di un metodo per pura analogia con fenomeni fisici e naturali.

Ipercomplicazione: introdurre componenti e parametri pletorici, come se potessero portare solo miglioramenti.

Number crunching: fare calcoli pesanti e sofisticati con numeri di dubbia utilità.

Overfitting: sovradattare componenti e parametri dell'algoritmo allo specifico insieme di dati usati nella valutazione sperimentale.

Integralismo euristico: usare euristiche quando esistono metodi esatti utilizzabili.

Effetto SUV: confidare nella potenza dell'hardware

2
Problemi tipici

2.1 Problemi su insiemi pesati: il problema dello zaino o Knapsack

Si vuole scegliere da un insieme di oggetti voluminosi un sottoinsieme di valore massimo che si possa racchiudere in uno zaino di capacità limitata.

Viene definito da 4 elementi significativi:

1. Un insieme O si oggetti elementari.
2. Una funzione $v : O \to \mathbb{N}$ che descrive la capacità di uno zaino.
3. Un numero $V \in \mathbb{N}$ che descrive la capacità di uno zaino.
4. Una funzione $\phi : O \to \mathbb{N}$ che descrive il valore di ogni oggetto.

Insieme base Il suo insieme base è banalmente il set degli oggetti:

$$B = 0$$

Regione ammissibile la sua regione ammissibile contiene i sottoinsiemi di oggetti di volume totale non superiore alla capacità dello zaino:

$$X = \left\{ x \subseteq B : \sum_{j \in x} v_j \leqslant V \right\}$$

Obbiettivo L'obbiettivo è massimizzare il valore complessivo degli oggetti scelti.

$$\max_{x \in X} f(x) = \sum_{j \in x} \phi_j$$

2.2 Problemi su insiemi dotati di metrica: il Maximum Diversity Problem

Si desidera scegliere da un insieme di punti un sottoinsieme di k punti con la massima somma delle distanze reciproche.

Viene definito da 3 elementi significativi:

1. Un insieme P di punti.
2. Una funzione $d : P \times P \to \mathbb{N}$ che dà la distanza fra coppie di punti.
3. Un numero $k \in \{1, \ldots, |P|\}$ che dà il numero di punti da scegliere.

Insieme base L'insieme coincide con l'insieme dei punti:

$$B = P$$

Regione ammissibile La regione ammissibile contiene i sottoinsiemi di k punti:

$$X = \{x \subseteq B : |x| = k\}$$

Obbiettivo L'obbiettivo è massimizzare la distanza totale reciproca fra i punti scelti.

$$\max_{x \in X} f(x) = \sum_{i,j \in x} d_{ij}$$

2.3 Problemi su insiemi da ripartire: Bin Packing Problem

Si vuole dividere un insieme di oggetti voluminosi nel minimo numero di contenitori di capacità data.

Viene definito da 4 elementi significativi:

1. Un insieme O di oggetti elementari.
2. Una funzione $v : O \to \mathbb{N}$ che descrive il volume di ogni oggetto.
3. Un insieme C di contenitori.
4. Un numero $V \in \mathbb{N}$ che dà il volume dei contenitori.

Insieme base L'insieme base contiene le coppie di oggetti e contenitori:

$$B = O \times C$$

Regione ammissibile La regione ammissibile contiene le partizioni degli oggetti tra i contenitori tali da non eccedere le capacità di alcun contenitore:

$$X = \left\{ x \subseteq B : |x \in B_o| = 1 \forall o \imath O, \sum_{(o,c) \in B^c} v_o \leqslant V \forall c \in C \right\}$$

con $B_o = \{(i,j) \in B : i = o\}$, $B^c = \{(i,j) \in B : j = c\}$.

Obbiettivo L'obbiettivo è minimizzare il numero di contenitori usati.

$$\min_{x \in X} f(x) = \left| \{c \in C; x \cap B^c \neq \emptyset\} \right|$$

2.4 Problemi su insiemi da ripartire: Parallel Machine Scheduling Problem

Si vuole dividere un insieme di lavorazioni fra un dato insieme di macchine minimizzando il tempo di completamento.

Viene definito da 3 elementi significativi:

1. Un insieme L di lavorazioni.
2. Una funzione $d : L \to \mathbb{N}$ che descrive la durata di ogni lavorazione.
3. Un insieme M di macchine.

Insieme base L'insieme base contiene le coppie tra lavorazioni e macchine:

$$B = L \times M$$

Regione ammissibile La regione ammissibile contiene le partizioni delle lavorazioni tra le macchine:

$$X = \{x \subseteq B : |x \cap B_l| = 1 \forall\, l \in L\}$$

Obbiettivo L'obbiettivo è minimizzare la massima durata totale per ogni macchina:

$$\min_{x \in X} f(x) = \max_{m \in M} \sum_{l:(l,m) \in x} d_l$$

2.5 Problemi su funzioni logiche: il max-SAT

Data una CNF (Conjunctive normal form) si cerca l'assegnamento di verità alle variabili logiche che soddisfi l'insieme di formule di peso e cardinalità massima.

Viene definito su 3 elementi significativi:

1. Un insieme V di variabili logiche x_j
2. Una forma congiuntiva normale (CNF) definita su tali variabili.
3. Una funzione peso w associata alle formule che compongono la CNF.

È importante ricordare che in questo contesto valgono le seguenti proprietà:

1. La variabile logica x_j è una variabile che assume valore in $B = \{0, 1\}$
2. Il letterale l_j è una funzione ridotta a una variabile affermata o negata:

$$l_j \in \{x_j, \overline{x}_j\}$$

3. La formula logica è una disgiunzione o somma logica (OR) di letterali:

$$C_i(x) = l_{i,1} \vee \ldots \vee l_{i,n_i}$$

4. La CNF è una congiunzione o prodotto logico (AND) di formule logiche:

$$CNF(x) = C_1 \wedge \ldots \wedge C_n$$

5. Soddisfare una funzione logica significa farle assumere valore 1.

Insieme base L'insieme base è l'insieme degli assegnamenti di verità:

$$B = V \times B = \{(x_1, 0), (x_1, 1), \ldots, (x_n, 0), (x_n, 1)\}$$

Regione ammissibile La regione ammissibile contiene i sottoinsiemi di assegnamenti:

Assegnamenti completi Per ogni variabile compare almeno un letterale.

Assegnamenti coerenti Per ogni variabile compare un solo letterale.

$$X = \{x \subseteq B : |x \cup B_v| = 1 \quad \forall v \in V\}$$

con $B_{x_j} = \{(x_j, 0), (x_j, 1)\}$.

Obbiettivo L'obbiettivo è massimizzare il peso totale delle formule soddisfatte:

$$\max_{x \in X} f(x) = \sum_{i:C_i(x)=1} w_i$$

2.6 Problemi su matrici numeriche

2.6.1 Set Covering

Data una matrice binaria e un vettore di costi associati alle colonne, si cerca il sottoinsieme di colonne di costo minimo che copra tutte le righe. Dire che "la colonna $j \in C$ copre la riga $i \in R$" significa che $a_{ij} = 1$

Viene definito da 2 elementi:

1. Una matrice binaria $A \in B^{m,n}$ costituita da un insieme di righe R e un insieme di colonne C.
2. Una funzione $c : C \to \mathbb{N}$ che indica il costo di ogni colonna.

Insieme base L'insieme base è l'insieme delle colonne:

$$B = C$$

Regione ammissibile La regione ammissibile contiene i sottoinsiemi di colonne che coprono tutte le righe:

$$X = \left\{ x \subseteq B : \sum_{j \in x} a_{ij} \geqslant 1 \quad \forall i \in R \right\}$$

Obbiettivo L'obbiettivo è minimizzare il costo totale delle colonne scelte:

$$\min_{x \in X} f(x) = \sum : j \in x c_j$$

2.6.2 Set Packing

Data una matrice binaria ed un vettore di valori associati alle colonne, si cerca il sottoinsieme di colonne di valore massimo che non confliggano. Dire che "Le colonne j' e j'' confliggono" significa che $a_{ij'} = a_{ij''} = 1$.

Viene definito da 2 elementi:

1. Una matrice binaria $A \in B^{m,n}$ costituita da un insieme di righe R e un insieme di colonne C.
2. Una funzione $\phi : C \to \mathbb{N}$ che indica il valore di ogni colonna,

Insieme base L'insieme base è l'insieme delle colonne:

$$B = C$$

Regione ammissibile La regione ammissibile contiene i sottoinsiemi di colonne che non confliggono:

$$X = \left\{ x \subseteq B : \sum_{j \in x} a_{ij} \leqslant 1 \quad \forall i \in R \right\}$$

Obbiettivo L'obbiettivo è massimizzare il valore totale delle colonne scelte:

$$\max_{x \in X} f(x) = \sum_{j \in x} \phi_j$$

2.6.3 Set partitioning

Data una matrice binaria e un vettore di costi associati alle colonne, si cerca il sottoinsieme di colonne di costo minimo che copra tutte le righe senza conflitti.

Viene definito da 2 elementi:

1. Una matrice binaria $A \in B^{m,n}$ costituita da un insieme di righe R e un insieme di colonne C.
2. Una funzione $c : C \to \mathbb{N}$ che indica il costo di ogni colonna.

Insieme base L'insieme base è l'insieme delle colonne:

$$B = C$$

Regione ammissibile La regione ammissibile contiene i sottoinsiemi di colonne che coprono tutte le righe e non confliggono:

$$X = \left\{ x \subseteq C : \sum_{j \in x} a_{ij} = 1 \quad \forall i \in R \right\}$$

Obbiettivo L'obbiettivo è minimizzare il costo totale delle colonne scelte:

$$\min_{x \in X} f(x) = \sum_{j \in x} c_j$$

2.7 Problemi su grafo

2.7.1 Vertex Cover

Dato un grafo non orientato $G = (V, E)$ si cerca il sottoinsieme di vertici di cardinalità minima tale che ogni lato del grafo vi incida.

Insieme base L'insieme base è dato dall'insiem dei vertici:

$$B = V$$

Regione ammissibile La regione ammissibile contiene i sottoinsiemi di vertici tali che tutti i lati del grafo vi incidono:

$$X = \{x \subseteq B : x \cap (i, j) \neq \emptyset \quad \forall (i, j) \in E\}$$

Obbiettivo L'obbiettivo consiste nel minimizzare il numero di vertici scelti.

$$\min_{x \in X} f(x) = |x|$$

2.7.2 Maximum Clique Problem

Dato un grafo non orientato e una funzione di peso definita sui vertici, si cerca il sottoinsieme di vertici fra loro adiacenti di peso massimo.

Viene caratterizzato da 2 elementi:

1. Un grafo non orientato $G = (V, E)$.
2. Una funzione $w : V \to \mathbb{N}$ che indica il peso di ogni vertice.

Insieme base L'insieme base è l'insieme dei vertici.

$$B = V$$

Regione ammissibile La regione ammissibile contiene i sottoinsiemi di vertici reciprocamente adiacenti.

$$X = \{x \subseteq B : (i, j) \in E \quad \forall i \in x, \quad \forall j \in x \setminus i\}$$

Obbiettivo L'obbiettivo è massimizzare il peso dei vertici scelti:

$$\max_{x \in X} f(x) 0 \sum_{j \in x} w_j$$

2.7.3 Maximum Indipendent Set Problem

Dato un grafo non orientato e una funzione peso definita sui vertici, si cerca il sottoinsieme di vertici fra loro non adiacenti di peso massimo.

Viene caratterizzato da 2 elementi:

1. Un grafo non orientato $G = (V, E)$.
2. Una funzione $w : V \to \mathbb{N}$ che indica il peso di ogni vertice.

Insieme base L'insieme base è l'insieme dei vertici:

$$B = V$$

Regione ammissibile La regione ammissibile contiene i sottoinsiemi di vertici reciprocamente non adiacenti:

$$X = \left\{ x \subseteq B : \left(i, j\right) \notin E \quad \forall i \in x, \quad \forall j \in x \setminus \{i\} \right\}$$

Obbiettivo L'obbiettivo è massimizzare il peso dei vertici scelti:

$$\max_{x \in X} f(x) = \sum_{j \in x} w_j$$

2.7.4 Travelling Salesman Problem

Dato un grafo orientato e una funzione costo definita sugli archi, si cerca il ciclo di costo minimo che ricopra tutti i nodi del grafo.

Viene caratterizzato da 2 elementi:

1. Un grafo orientato $G = (N, A)$.
2. Una funzione $c : A \to \mathbb{N}$ che indica il costo di ogni lato.

Insieme base L'insieme base è l'insieme degli archi:

$$B = A$$

Regione ammissibile La regione ammissibile contiene i cicli che coprono tutti i nodi del grafo, cioè i cicli hamiltoniani.

Obbiettivo L'obbiettivo è minimizzare il costo degli archi scelti.

$$\min_{x \in X} f(x) = \sum_{j \in x} c_j$$

2.7.5 Capacitated Spanning Tree Problem

Dato un grafo non orientato, un vertice radice, una funzione costo definita sui lati, una definita sui vertici ed una capacità massima, si cerca l'albero ricoprente di costo minimo tale che ogni sotto-albero appeso alla radice abbia peso non superiore alla capacità data.

Viene caratterizzato da 4 elementi:

1. Un grafo non orientato $G = (V, E)$ con un vertice radice $r \in V$.
2. Una funzione $c : E \to \mathbb{N}$ che indica il costo di ogni lato.
3. Una funzione $w : V \to \mathbb{N}$ che indica il peso di ogni vertice.
4. Un numero $W \in \mathbb{N}$ che indica la capacità di ogni sotto-albero.

Insieme base L'insieme base è l'insieme dei lati:

$$B = E$$

Regione ammissibile La regione ammissibile contiene gli alberi ricoprenti tali che il peso dei vertici ricoperti da ogni sotto-albero appeso alla radice non superi W. È importante tenere a mente che il test di ammissibilità richiede la visita del grafo.

Obbiettivo L'obbiettivo è di minimizzare il costo dei lati scelti.

$$\min_{x \in X} f(x) = \sum_{j \in x} c_j$$

Descrizione alternativa

Se si procede ad aggiungere un insieme di sotto-alberi T di cardinalità $|V| - 1$, alcuni anche vuoti, si possono ridefinire insieme base, regione ammissibile e obbiettivo come segue:

Insieme base L'insieme base è l'insieme di coppie (vertice, sotto-albero):

$$B = V \times T$$

Regione ammissibile La regione ammissibile contiene le partizioni dei vertici in sottoinsiemi di peso $\leqslant W$ connessi (richiede chiaramente la visita su grafo completo).

Obbiettivo L'obbiettivo consiste nel minimizzare la somma dei costi degli alberi che ricoprono ciascun sotto-insieme di vertici più i lati che li collegano alla radice.

2.7.6 Vehicle Routing Problem

Dato un grafo orientato, un nodo deposito, una funzione costo definita sugli archi, una definita sui nodi ed una capacità, si cerca l'insieme di cicli di costo minimo che passano per il deposito e hanno ciascuno peso totale non superiore alla capacità.

Viene caratterizzato da 4 elementi:

1. Un grafo orientato $G = (N, A)$ con un nodo deposito $d \in N$.
2. Una funzione $c : A \to \mathbb{N}$ che indica il costo di ogni arco.
3. Una funzione $w : N \to \mathbb{N}$ che indica il peso di ogni nodo.
4. Un numero $W \in \mathbb{N}$ che indica la capacità di ogni ciclo.

Insieme Base L'insieme base potrebbe essere:

Insieme degli archi $B = A$

Insieme delle coppie $B = N \times C$

Regione ammissibile La regione ammissibile potrebbe contenere:

1. Gli insiemi di archi che coprono tutti i nodi con cicli passanti per il deposito e di peso non superiore a W, che richiede ancora la visita del grafo.
2. Le partizioni dei nodi in sottoinsiemi di peso ciascuno non superiore a W e copribile con un ciclo.

Obbiettivo L'obbiettivo è minimizzare il costo degli archi scelti:

$$\min_{x \in X} f(x) = \sum_{j \in x} c_j$$

2.8 Relazioni tra i problemi

2.8.1 Relazione tra Maximum Clique Problem e Maximum Indipendent Set Problem

Dato un grafo $G = (V, E)$ si costruisce il grafo complementare $\overline{G} = (V, (V \times V) \setminus E)$. Si trova quindi la soluzione ottima del MISP su $\overline{G}$. I vertici corrispondenti danno una soluzione ottima del MCP su G.

Relazione tra Vertex Cover e Set Partitioning

Ogni istanza di Vertex Cover può essere tradotta in un'istanza di Set Partitioning: ogni lato i corrisponde a una riga della matrice di copertura A, ogni vertice j corrisponde a una colonna di A.

Se il lato i incide nel vertice j, si pone $a_{ij} = 1$, altrimenti $a_{ij} = 0$.

La soluzione ottima del Set Partitioning dà la soluzione ottima nel Vertex Cover.

Relazione tra Bin Packing Problem e Parallel Machine Scheduling Problem

I due problemi sono equivalenti: le lavorazioni corrispondono agli oggetti e le macchine ai contenitori, ma vi sono due differenze importanti da tenere a mente:

1. Nel Bin Packing è data la capacità e si minimizza il numero di contenitori.
2. Nel Parallel Machine Scheduling è dato il numero di macchine, e si minimizza il tempo di completamento.

Per minimizzare il numero di contenitori del Bin Packing con una data capacità si procedere facendo un'ipotesi sul valore ottimo, quindi si costruisce il Parallel Machine Scheduling corrispondente e si valuta il tempo di completamento ottimo. Se questo eccede la capacità si va ad aumentare l'ipotesi iniziale e se non la eccede si prova a ridurla.

3

Analisi teorica di prestazioni

Si occupa di dimostrare che l'algoritmo fornisce soluzioni con una data garanzia di qualità, sempre o con una data frequenza.

Chiamiamo la soluzione euristica $f_A(l)$ e quella ottima $f^*(l)$:

Definizione 3.0.1 (Differenza assoluta).

$$\tilde{\delta}_A(l) = \left| f_A(l) - f^*(l) \right| \geqslant 0$$

Dipende dall'unità di misura dell'obbiettivo e pertanto si usa di rado.

Definizione 3.0.2 (Differenza relativa).

$$\delta_A(l) = \frac{\left| f_A(l) - f^*(l) \right|}{f^*(l)} \geqslant 0$$

È frequente nell'analisi sperimentale.

Definizione 3.0.3 (Rapporto di approssimazione).

$$\rho_A(l) = \max\left[\frac{f_A(l)}{f^*(l)}, \frac{f^*(l)}{f_A(l)} \right] \geqslant 1$$

È frequente nell'analisi teorica.

3.1 Analisi teorica nel caso pessimo

Il fattore α_A $(\overline{\alpha}_A)$ si dice garanzia di approssimazione relativa (assoluta.)

In generale, la garanzia dipende dalla dimensione dell'istanza:

$$\rho_A(I) \leqslant \alpha_A(n) \quad \forall I \in \mathbb{I}_n, n \in \mathbb{N}$$

Contrariamente all'efficacia, per l'efficienza potrebbe anche non esserne dipendente.

Definizione 3.1.1 (Approssimazione assoluta).

$$\exists \overline{\alpha}_A \in \mathbb{N} : \overline{\delta}_A(l) \leqslant \overline{\alpha}_A \quad \forall I \in \mathbb{I}$$

Un esempio (raro) è l'algoritmo di Vizing per l'Edge Coloring $(\overline{\alpha} = 1)$.

Definizione 3.1.2 (Approssimazione relativa).

$$\exists \alpha_A \in \mathbb{R}^+ : \rho_A(I) \leqslant \alpha_A \quad \forall I \in \mathbb{I}$$

3.1.1 Ricavare la garanzia di approssimazione

Per un problema di minimizzazione si vuole dimostrare che $f_A(I) \leqslant \alpha f^*(I) \quad \forall I \in \mathbb{I}$.

1. Si trova un modo per costruire una **stima per difetto** $LB(I)$:
$$LB(I) \leqslant f^*(I) \quad I \in \mathbb{I}$$
2. Si trova un modo per costruire una stima per eccesso $UB(I)$, che sia legata a $LB(I)$ da un coefficiente α, o da una funzione $\alpha(n)$:
$$UB(I) = \alpha LB(I) \quad I \in \mathbb{I}$$
3. Si trova un algoritmo A la cui soluzione non è peggiore di $UB(I)$:
$$f_A(I) \leqslant UB(I) \quad I \in \mathbb{I}$$
4. Quindi $f_A(I) \leqslant UB(I) = \alpha LB(I) \leqslant \alpha f^*(I) \quad \forall I \in \mathbb{I}$:
$$f_A(I) \leqslant \alpha f^*(I) \quad \forall I \in \mathbb{I}$$

3.2 Algoritmo 2-approssimato per il VCP

Dato un grafo non orientato $G = (V, E)$ si cerca il sottoinsieme di vertici di cardinalità minima tale che ogni lato del grafo vi incida.

Definizione 3.2.1 (Matching). Si dice **matching** un insieme di lati non adiacenti fra loro.

Definizione 3.2.2 (Matching massimale). Un matching viene detto massimale quando è tale che qualsiasi altro lato del grafo è adiacente a un lato del matching.

3.2.1 Algoritmo del matching

1. Si costruisce un matching massimale $M \subseteq E$ cioè tale che ogni altro lato di E è adiacente a un lato di M.
2. L'insieme dei vertici estremi dei lati del matching è una soluzione, che può essere migliorata eliminando dei vertici ridondanti:
$$x_A = \bigcup_{(u,v) \in M} \{u, v\}$$

Dimostrazione 3.2.3 (L'algoritmo del matching è 2-approssimato).
1. La cardinalità del matching M è una stima per difetto $LB(I)$.
   (a) La cardinalità di una copertura ottima per qualsiasi sottoinsieme di lati $E' \subseteq E$ non supera quella di una copertura ottima per E (coprire tutti i lati costa di più che coprire solo i lati del matching):
   $$\left|x^*_{E'}\right| \leqslant \left|x^*_E\right|$$
   (b) La copertura ottima di un matching M ha cardinalità $|M|$, cioè per ogni lato del matching basta e occorre un vertice diverso.
2. Includendo entrambi i vertici di gni lato nel matching si ottiene una stima per eccesso che copre sia il matching, sia i lati adiacenti, che risulta di valore $UB(I) = 2LB(I)$, cioè vi sono due vertici diversi per ogni lato.
3. L'algoritmo del matching dà soluzioni di valore $f_A(I) \leqslant UB(I)$ (scremando i vertici ridondanto se ve ne sono).

Ne deriva che $f_A(I) \leqslant 2f^*(I) \quad \forall I \in \mathbb{I}$ cioè $\alpha_A = 2$. □

3.3 Schemi di approssimazione

Definizione 3.3.1 (Schema di approssimazione). Uno **Schema di Approssimazione** è un algoritmo A parametrico con α a piacere:

$$T_{A_\alpha} \in O\big(f(n,\alpha)\big) \quad \alpha \in [1;+\infty)$$

Uno schema di approssimazione può essere:

Polinomiale se $f(n,\alpha)$ è un polinomio in n $\quad \forall \alpha$ fissato.

Pienamente polinomiale se $f(n,\alpha)$ è un polinomio in n e in $\frac{1}{\alpha}$

È importante tenere a mente che esistono **problemi non approssimabili**, cioè problemi per cui l'unico algoritmo approssimato è un algoritmo esatto: un esempio è il TSP, che contiene istanze non approssimabili.

3.3.1 Approssimazione del TSP: Algoritmo del doppio albero

In alcune istanze del TSP è lecito considerare:

1. Il grafo $G = (N, A)$ completo.
2. La funzione c sia una distanza valida: cioè goda di simmetria e disuguaglianza triangolare.

Definizione 3.3.2 (Algoritmo del doppio albero).

1. Consideriamo il grafo completo non orientato corrispondente a G.
2. Costruiamo un albero ricoprente di costo minimo $T^* = (N, X^*)$.
3. L'insieme x degli archi di G corrispondenti ai lati di X^* forma un ciclo orientato che passa per ogni nodo, in generale più volte.
4. Finché ci sono nodi con più di un arco uscente ed entrante:
   (a) Si sceglie un nodo qualsiasi j e una coppia di archi (i,j) e (j,k) in x'.
   (b) Si sostituisce la coppia di archi con l'arco diretto.

$$x = x \setminus \{(i,j),(j,k)\} \cup \{(i,k)\}$$

Dimostrazione 3.3.3 (L'algoritmo del doppio albero è 2-approssimato).

1. Il costo dell'albero ricoprente minimo è una stima per difetto $LB(I)$: togliendo un arco a un ciclo hamiltoniano si ottiene un cammino hamiltoniano meno costoso e un cammino hamiltoniano è un particolare tipo di albero ricoprente.
2. Sostituendo ogni lato dell'albero con due archi opposti si ottiene una stima per eccesso, essendo un cammino hamiltoniano, di valore $UB(I) = 2LB(I)$, cioè due archi sostituiscono ogni lato.
3. L'algoritmo del doppio albero dà soluzioni di valore $f_A(I) \leqslant UB(I)$, cioè sostituendo due archi consecutivi con uno diretto il costo cala.

Ne deriva che $f_A(I) \leqslant 2f^*(I) \forall I \in \mathbb{I}$, cioè $\alpha_A = 2$. □

3.4 Algoritmi approssimati randomizzati

Definizione 3.4.1 (Algoritmo randomizzato approssimato). L'algoritmo esegue operazioni che non dipendono solo dai dati, ma anche da numeri pseudocasuali: per un tale algoritmo A $f_A(I)$ e $\rho_A(I)$ sono variabili aleatorie.

Un **algoritmo randomizzato approssimato** ha un rapporto di approssimazione il cui valore atteso è limitato da una costante:

$$\mathbb{E}\left(\rho_A(I)\right) \leqslant \alpha_A \quad \forall I \in \mathbb{I}$$

3.4.1 Esempio: determinare l'approssimazione randomizzata per Max-SAT

Utilizzando un approccio puramente casuale, che assegna il valore di verità con probabilità $\frac{1}{2}$, procediamo a determinare il valore atteso della soluzione:

Sia $C_x \subseteq \{1,\ldots,m\}$ l'insieme delle formule soddisfatte dalla soluzione x.

Il valore dell'obbiettivo $f(x)$ è il peso totale delle formule in C_x, mentre il valore atteso rispetto a tutte le soluzioni x proposte dall'algoritmo A é:

$$\mathbb{E}\left(f_A(I)\right) = \mathbb{E}\left(\sum_{i \in C_x} w_i\right) = \sum_{i \in C}\left(w_i \cdot \mathbb{P}\left(i \in C_x\right)\right)$$

Sia k_i il numero di letterali della formula $i \in C$ e $k_{min} = \min_{i \in C} k_i$:

$$\mathbb{P}\left(i \in C_x\right) = 1 - \left(\frac{1}{2}\right)^{k_i} \geqslant 1 - \left(\frac{1}{2}\right)^{k_{min}} \quad \forall i \in C$$

Ne segue che:

$$\mathbb{E}\left(f_A(I)\right) \geqslant \sum_{i \in C} w_i \cdot \left[1 - \left(\frac{1}{2}\right)^{k_{min}}\right] = \left[1 - \left(\frac{1}{2}\right)^{k_{min}}\right] \sum_{i \in C} w_i$$

E siccome:

$$f^*(I) \leqslant \sum_{i \in C} w_i \quad \forall I \in \mathbb{I} \quad \text{e} \mathbb{E}\left(\rho_A(I)\right) = \frac{f^*(I)}{\mathbb{E}\left(f_A(I)\right)}$$

Si ottiene:

$$\mathbb{E}\left(\rho_A(I)\right) \leqslant \frac{1}{\left[1 - \left(\frac{1}{2}\right)^{k_{min}}\right]} \leqslant 2$$

4

Analisi sperimentale

L'analisi sperimentale indaga le leggi che regolano il comportamento degli algoritmi:

Si tratta di:

1. Ottenere indici di efficacia e di efficienza di un algoritmo.
2. Descrivere il legame fra gli indici di efficacia o efficienza e valori parametrici delle istanze.
3. Confrontare l'efficacia di algoritmi diversi per indicare il migliore.
4. Suggerire miglioramenti all'algoritmo.

4.1 Benchmark

Non potendo testare tutte le istanze, occorre definire un campione.

Un campione significativo deve rappresentare:

Diverse dimensioni cosa particolarmente importante per l'analisi sperimentale della complessità.

Caratteristiche strutturali come è "fatto" il modello dei dati.

Tipologie di applicazione logistica, telecomunicazioni, produzione...

Tipologie di generazione realistiche, artificiali, trasformazioni di altri problemi...

Tipologie di distribuzione probabilistica uniforme, normale, esponenziale...

Tipicamente si cerca di concentrarsi sulle istanze **più frequenti in pratica**, in particolare quelle **più difficili** ma evitando quelle che **richiedono troppo tempo**.

Considerando l'esecuzione di un algoritmo A come un esperimento casuale: un'analisi significativa è possibile solo tenendo fissa la dimensione n (con gli altri parametri rilevanti suggeriti dall'analisi del caso pessimo).

In particolare:

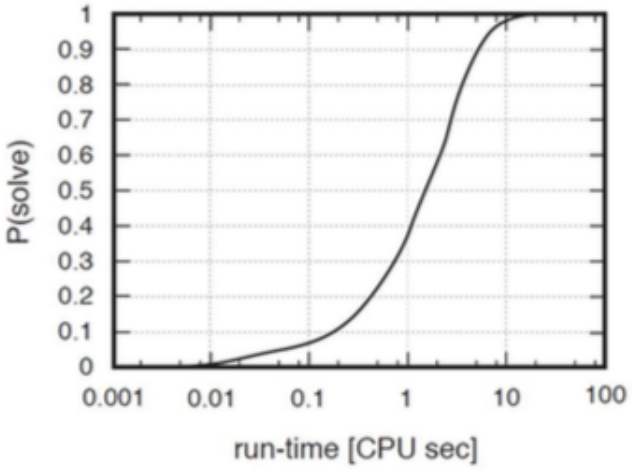
1. L'insieme delle istanze $\mathscr{I}_n$ costituisce lo **spazio degli esiti**.
2. Il tempo di calcolo $T_A(I)$ è la variabile aleatoria sotto indagine.

Le prestazioni A sono descritte dalle proprietà statistiche di $T_A(I)$, cioè il suo valore atteso e varianza in $\mathscr{I}_n$.

Per il algoritmi polinomiali, se i parametri rilevanti sono fissati, la media tende ad essere un indice affidabile.

4.2 Diagramma RTD

La funzione di distribuzione della variabile aleatoria $T_A(I)$ in $\mathscr{I}_n$ fornisce il diagramma della **Run Time Distribution (RTD)**.

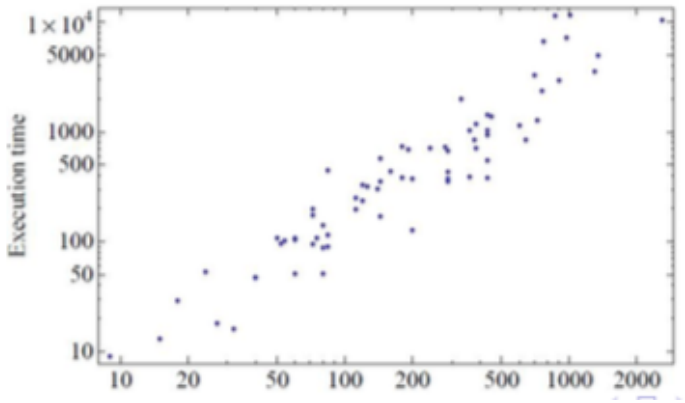


Quando tutti i parametri influenti sono stati individuati il diagramma **RTD** degenera in un gradino, vale a dire che il tempo è quasi uguale per tutte le istanze di $\mathscr{I}_n$.

Se è molto distribuito potrebbero esistere altri parametri interessanti.

4.3 Diagramma di Scaling

La complessità asintotica indica per il caso pessimo una fascia di andamenti di ampiezza imprecisata.



Lo studio empirico al variare di n fornisce quindi le informazioni mancanti.

Per eseguirlo, bisogna:

1. Estrarre una sequenza di campioni $\mathscr{I}_n$ per valori crescenti di n.
2. Applicare l'algoritmo e registrare i tempi $T_A(I)$
3. Tracciare la nuvola dei punti $(n, T(I))$ o delle medie $\left(n, \frac{\sum_{I \in \mathscr{I}_n} T(I)}{|\mathscr{I}_n|}\right)$
4. Ipotizzare una famiglia di interpolanti con l'aiuto dell'analisi teorica
5. Tarare i parametri dell'interpolante.

4.3.1 Interpolazione

Si procede per tentativi per capire su quale tipo di scala si genera un diagramma di scaling lineare:

Scala lineare, su ascisse e ordinate La funzione interpolante è lineare.

$$T(n) = \alpha n + \beta \Leftrightarrow c_1 n + c_2$$

Scala logaritmica, su ascisse e ordinate La funzione interpolante è polinomiale.

$$\log T(n) = \alpha \log n + \beta \Leftrightarrow 2^{\beta} n^{\alpha}$$

Scala semi-logaritmica, lineare su ascisse e logaritmica su ordinate La funzione interpolante è esponenziale.

$$\log T(n) = \alpha n + \beta \Leftrightarrow 2^{\beta} \left(2^{\alpha}\right)^{n}$$

4.4 Analisi dell'efficacia di un algoritmo

Considerando l'esecuzione di un algoritmo A come un esperimento casuale:

1. L'insieme delle istanze $\mathscr{I}_n$ costituisce lo **spazio degli esiti**.
2. La differenza relativa $\delta_A(I)$ è la variabile aleatoria sotto indagine.

Le prestazioni di A sono descritte dalle proprietà statistiche di $\delta_A(I)$.

4.4.1 Solution Quality Distribution (SQD)

Una descrizione completa di $\delta_A(I)$ è data dalla funzione di distribuzione:

$$F_A(\alpha) = \mathbb{P}(\delta_A(I) \leqslant \alpha) \quad \forall \alpha \in \mathbb{R}$$

Il suo andamento è il diagramma SQD.

Per un qualsiasi algoritmo, la funzione di distribuzione di $\delta_A(I)$ è **monotona non decrescente**, **nulla per** $\alpha < 0$ e tenda a 1 per $\alpha \to +\infty$.

In particolare:

Per algoritmi esatti è una funzione a gradino, pari a 1 $\quad \forall \alpha \geqslant 0$

Per algoritmi $\overline{\alpha}$-approssimati è una funzione pari a 1 $\quad \forall \alpha \geqslant \overline{\alpha}$

Come generare il diagramma SQD

In pratica si ricostruisce un **diagramma campionario** della funzione:

1. Si genera un campione $\overline{\mathscr{I}} \subset \mathscr{I}$ abbastanza grande di istanze indipendenti.
2. Si applica l'algoritmo a ogni istanza $I \in \overline{\mathscr{I}}$ e si costruisce l'insieme:

$$\Delta_A\left(\overline{\mathscr{I}}\right) = \left\{\delta_A(I) : I \in \overline{\mathscr{I}}\right\}$$

3. Si ordina per valori non decrescenti.
4. SI riportano nel grafico i punti $\left(\delta_j, \frac{j}{\left|\overline{\mathscr{I}}\right|}\right) \quad \forall j \in 1,\ldots,\left|\overline{\mathscr{I}}\right|$

Spesso vengono realizzati numerosi diagrammi al variare di qualche parametro di interesse per l'algoritmo.

4.5 Test di Wilcoxon

Il test di Wilcoxon valuta la differenza fra i risultati dei due algoritmi $\delta_{A_1}(I) - \delta_{A_2}(I)$:

1. Si formula l'ipotesi nulla H_0: la differenza ha mediana nulla.
2. Si estrae un campione di istanze $\overline{\mathscr{I}}$ e gli si applicano i due algoritmi, ottenendo un campione di coppie di valori $(\delta_{A_1}, \delta_{A_2})$.
3. Si calcola la probabilità p di ottenere il risultato osservato o uno più estremo, supponendo vera l'ipotesi H_0.
4. Si sceglie un livello di significatività $\overline{p}$, tipicamente 5% o 1%, pari alla probabilità massima accettabile di rifiutare H_0 qualora fosse vera.
5. Se $p \leqslant \overline{p}$ si rifiuta H_0.

Il test non fa ipotesi sulla distribuzione di probabilità dei valori testati, quindi è un test non parametrico e ciò lo rende adatto a valutare le prestazioni di algoritmi euristici, dato che la distribuzione della differenza relativa non è nota.

Fa però 3 ipotesi:

1. I dati sono coppie di valori riferiti alla stessa popolazione.
2. Ogni coppia di dati è estratta indipendentemente dalle altre.
3. I dati sono misurate su scale almeno ordinali.

4.5.1 Esecuzione del test

1. Calcola le differenze assolute $\left|\delta_{A_1}(I) - \delta_{A_2}(I)\right| \quad \forall i \in 1, \ldots, \left|\overline{\mathscr{I}}\right|$.
2. Le ordina per valori crescenti e attribuisce un rango R_i a ciascuna.
3. Somma separatamente i ranghi delle coppie con differenza positiva e quelli delle coppie con differenza negativa.

$$\begin{cases} W^+ = \sum_{i:\delta_{A_1}(I_i) > \delta_{A_2}(I_i)} R_i \\ W^- = \sum_{i:\delta_{A_1}(I_i) < \delta_{A_2}(I_i)} R_i \end{cases}$$

   Se l'ipotesi nulla H_0 fosse vera, le due somme dovrebbero coincidere.

4. La differenza fra W^+ e W^- consente di calcolare il valore di p: dati i $2^{\left|\overline{I}\right|}$ esiti, corrispondenti a $\left|\overline{\mathscr{I}}\right|$ differenze positive o negative, si contano quelli con $\left|W^+ - W^-\right|$ pari o superiori all'osservazione.
5. Se $p < \overline{p}$, la differenza è significativa e un algoritmo prevale sull'altro:

   Caso $W^+ < W^-$**:** significa che A_1 è migliore di A_2.

   Caso $W^+ > W^-$**:** significa che A_1 è peggiore di A_2.

Il test di Wilcoxon può suggerire o che uno dei due algoritmi sia significativamente migliore dell'altro o che i due algoritmi sono statisticamente equivalenti.

Se il campione è composto da classi di istanze parametricamente distinte, algoritmi complessivamente equivalenti potrebbero non esserlo sulle singole istanze.

È sempre bene tenere a mente però che un algoritmo è migliore di un altro quando contemporaneamente ottiene risultati migliori e richiede un tempo inferiore.

4.6 Classificazione in base a tempo e qualità

Algoritmi completi o esatti: Per ogni istanza $I \in \mathscr{I}$ trovano l'ottimo in tempo finito.

Algoritmi probabilisticamente approssimativamente completi: Per ogni istanza $I \in \mathscr{I}$ la probabilità che trovino l'ottimo tende a 1 per $t \to +\infty$.

Algoritmi essenzialmente incompleti: esistono istanze per cui la probabilità di trovare l'ottima rimane sempre < 1 per $t \to +\infty$. Ne sono un esempio gli algoritmi greedy.

4.6.1 Generalizzazione a soluzioni approssimate

Un'ovvia generalizzazione sostituisce la ricerca dell'ottimo con quella do un dato livello di approssimazione:

Algoritmi α-completi: Per ogni istanza $I \in \mathscr{I}$ trovano una soluzione α-approssimata in tempo finito.

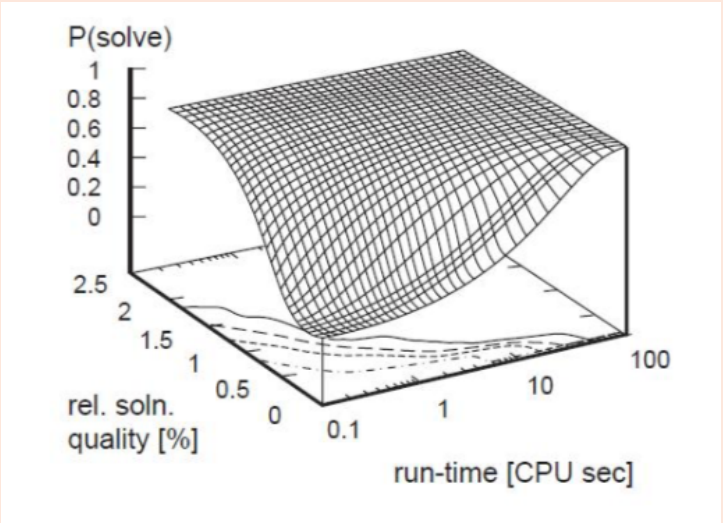
Algoritmi probabilisticamente approssimativamente α-completi: Per ogni istanza $I \in \mathscr{I}$ la probabilità che trovino una soluzione α-approssimata tende a 1 per $t \to +\infty$.

Algoritmi essenzialmente α-incompleti: esistono istanze per cui la probabilità di trovare una soluzione α-approssimata rimane sempre < 1 per $t \to +\infty$.

4.7 Probabilità di successo, diagrammi QRTD, SQD e SQT

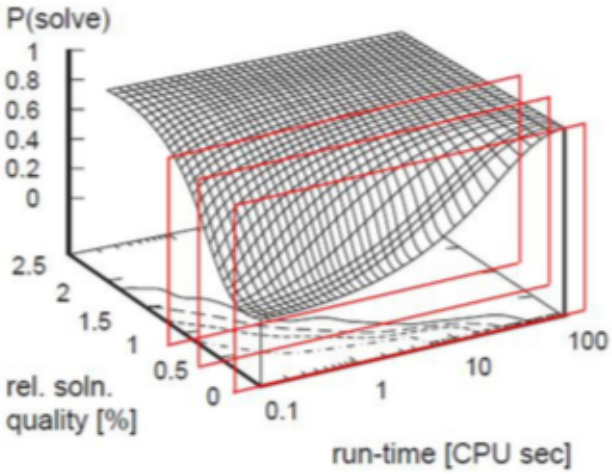
Definizione 4.7.1 (Probabilità di successo). Definiamo probabilità di successo $\pi_{A,n}(\alpha, t)$ la probabilità che l'algoritmo A trovi in tempo $\leqslant t$ una soluzione di livello $\leqslant \alpha$:

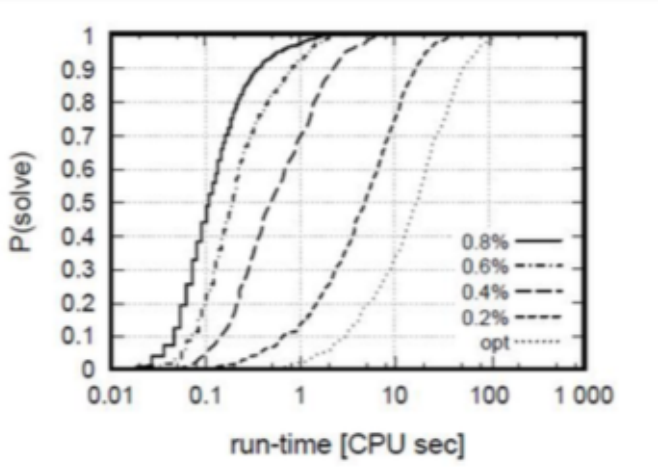
$$\pi_{A,n}(\alpha, t) = \mathbb{P}(\delta_A I, t \leqslant \alpha \,|\, I \in \mathscr{I}_n|)$$



4.7.1 I diagrammi Qualified Run Time Distribution (QRTD)

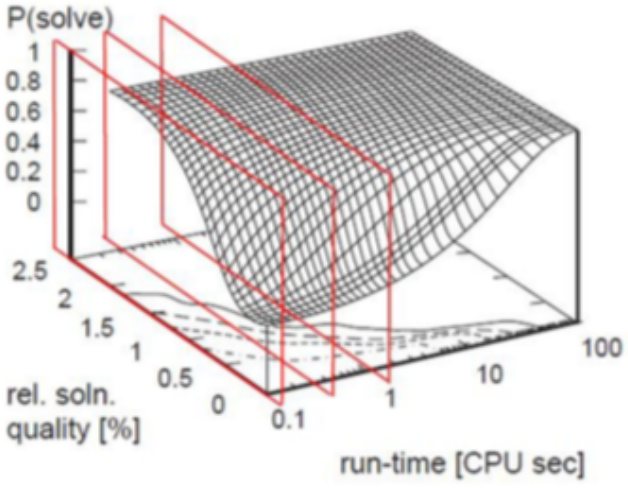
I diagrammi QRTD descrivono l'andamento del tempo necessario a raggiungere dei prefissati livelli di qualità e sono utili quando il tempo di calcolo non è una risorsa stringente.

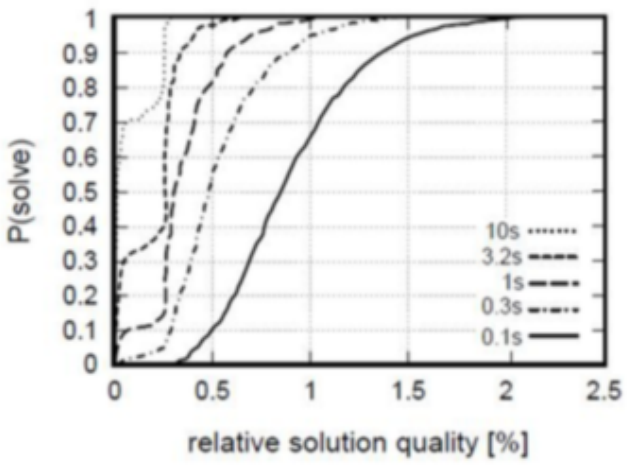




4.7.2 I Diagrammi Solution Quality Distribution (SQD)

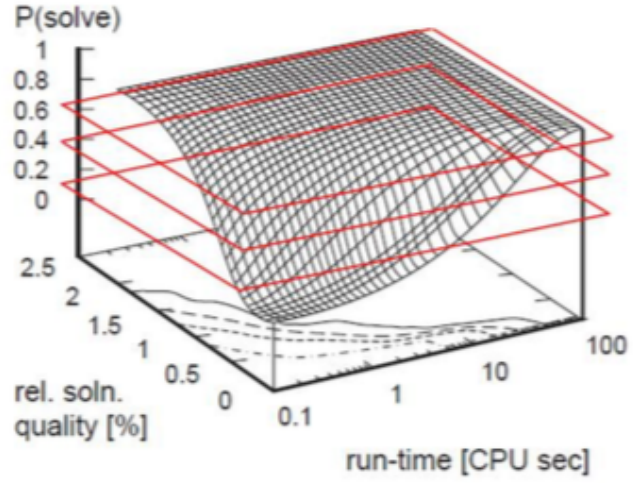
I diagrammi SQD descrivono l'andamento del livello di qualità raggiunto in un dato tempo di calcolo e sono utili quando il tempo do calcolo è una risorsa stringente.

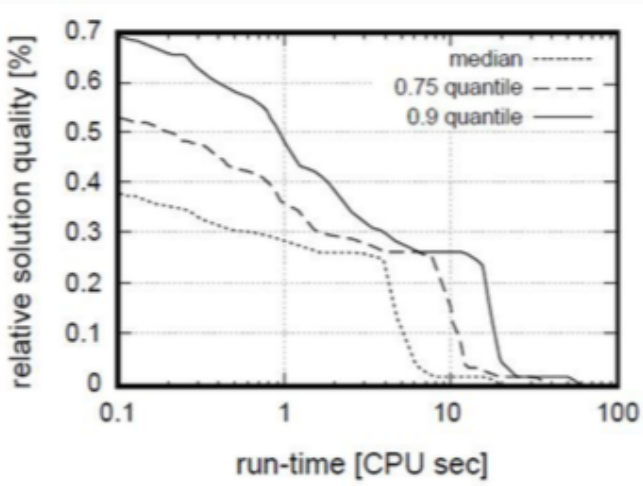




4.7.3 I diagrammi Solution Quality statistics over Time (SQT)

Sono diagrammi rappresentanti le linee di livello associate ai diversi quantili, che descrivono un compromesso fra qualità e tempo di calcolo. Un algoritmo robusto avrà linee molto ravvicinate tra loro.





5

Algoritmi costruttivi

Un algoritmo costruttivo A si definisce:

Puro se la funzione di scelta ϕ_A dipende solo dal nuovo elemento i.

Adattativo se ϕ_A dipende anche dalla soluzione corrente x.

Definizione 5.0.1 (Euristica costruttiva). Un'euristica costruttiva aggiorna passo per passo un sotto-insieme $x^{(t)}$:

1. Parte da un sotto-insieme vuoto:
$$x^0 = \emptyset$$
2. Si ferma se vale una condizione di fine opportunamente definita (tipicamente che i sotto-insiemi successivi non possono contenere soluzioni ottime).
3. Ad ogni passo t, sceglie l'elemento $i^{(t)} \in B$ "migliore" fra quelli ammissibili in base ad un opportuno criterio di scelta.
4. SI aggiunge $i^{(t)}$ al sottoinsieme corrente:
$$x^{(t+1)} = x^{(t)} \cup \{i^{(t)}\}$$
E non si torna più indietro sulla scelta fatta.
5. Si torna al punto 2.

Definizione 5.0.2 (Spazio di ricerca di un algoritmo costruttivo). Lo spazio di ricerca $\mathscr{F}_A$ di un algoritmo costruttivo A è la collezione dei sottoinsiemi che l'algoritmo considera validi e include:

1. Il sotto-insieme vuoto: $\emptyset \in \mathscr{F}_A$
2. Alcune soluzioni parziali, sottoinsiemi di soluzioni ammissibili.
3. Le soluzioni ammissibili promettenti, cioè quelle che non sono ancora dominate.

Definizione 5.0.3 (Grafo di costruzione). Il grafo di costruzione di un algoritmo costruttivo A ammette:

1. Come nodi i sottoinsiemi validi $x \in \mathscr{F}_A$.
2. Come archi le coppie di sotto-insiemi validi in cui il secondo ha un elemento in più del primo:
$$(x, x \cup \{i\}) : x \in \mathscr{F}_A,\ i \in B \setminus x \quad \text{e} \quad x \cup \{i\} \in \mathscr{F}_A$$

Si tratta di un grafo **aciclico** in cui ogni cammino massimale describe una possibile esecuzione dell'algoritmo

L'algoritmo A indica un insieme di estensioni ammissibili:

$$\text{Ext}_A(x) = \{i \in B \setminus x : x \cup \{i\} \in \mathscr{F}_A\} \quad \forall x \in \mathscr{F}_A$$

Occorre definire $\mathscr{F}_A$ in modo che:

1. Il test di appartenenza $x^{(t)} \in \mathscr{F}_A$ sia efficiente.
2. $\mathscr{F}_A$ includa (per quanto possibile) solo sottoinsiemi di soluzioni ammissibili, e magari ottime.

Un'euristica costruttiva A termina quando aggiungere qualsiasi elemento i al sotto-insieme corrente $x^{(t)}$ lo fa uscire dallo spazio di ricerca $\mathscr{F}_A$:

$$x^{(t)} \cup \{i\} \notin \mathscr{F}_A \quad \forall i \in B \setminus x^{(t)} \Rightarrow \text{STOP}$$

Dato che $\text{Ext}_A(x) = \{i \in B \setminus x : x \cup \{i\} \in \mathscr{F}_A\}$, la condizione diventa:

$$\text{Ext}_A\left(x^{(t)}\right) = \emptyset \Rightarrow STOP$$

Un algoritmo costruttivo A trova l'ottimo quando ad ogni passo t il sotto-insieme corrente $x^{(t)}$ è contenuto in almeno una soluzione ottima. Questa proprietà vale al primo passo, ma spesso si perde in qualche passo t.

Un'euristica costruttiva esegue al massimo $n = |B|$ passi.

Complessità computazionale 5.0.4 (Complessità dell'euristiche costruttive). La complessità di ogni passo è data da:

1. La costruzione di $\text{Etx}_A(x)$, in tempo $T_{\text{Ext}_A}(n)$.
2. La valutazione di $\phi_A(i,x) \quad \forall i \in \text{Ext}_A(x)$, in tempo $T_{\phi_A}(n)$.
3. L'estrazione del valore minimo e del corrispondente i.
4. L'aggiornamento di x.

In generale, è una complessità polinomiale di ordine piuttosto basso, in cui prevalgono le prime due componenti:

$$T_A(n) \in O\left(n\left(T_{\text{Ext}_A}(n) + T_{\phi_A}(n)\right)\right)$$

5.1 Matroidi e Greedoidi

Definizione 5.1.1 (Matroide). Un matroide è un sistema di insiemi $(B, \mathscr{F})$ con $\mathscr{F} eq 2^B$ tale che:

Assioma banale: $\emptyset \in \mathscr{F}$

Assioma di ereditarietà: se $x \in \mathscr{F}$ e $y \subset x$ allora $y \in \mathscr{F}$: vale a dire che ogni sottoinsieme valido si può costruire aggiungendo gli elementi in un ordine qualsiasi.

Assioma di scambio: $\forall x, y \in \mathscr{F} : |x| = \left|y\right| + 1, \exists i \in x \setminus y : y \cup \{i\} \in \mathscr{F}$, vale a dire che ogni sottoinsieme valido si può estendere con un opportuno elemento di qualsiasi altro sottoinsieme di cardinalità superiore.

Definizione 5.1.2 (Greedoide). Un greedoide è un sistema di insiemi $(B, \mathscr{F}) : \mathscr{F} \subseteq 2^B$ che rispetta l'assioma **banale** e **di scambio** dei matroidi ed una versione più debole dell'assioma di **ereditarietà** detto **assioma di accessibilità**:

$$x \in \mathscr{F} \ \wedge \ x \neq \emptyset \Rightarrow \exists i \in x : x \setminus \{i\} \in \mathscr{F}$$

Vale a dire che ogni sottoinsieme valido si può costruire aggiungendo gli elementi in un ordine opportuno.

5.2 Bestiario degli algoritmi costruttivi

5.2.1 Algoritmo First-Fit

Si parte con un sotto-insieme vuoto $x^{(0)} = \emptyset$ e si prende un oggetto i qualsiasi, quindi si sceglie il contenitore j in modo da minimizzare il numero di contenitori usati, scelta che dipende da x e non solamente da i.

Per fare questo, si prende il primo contenitore usato con capacità residua sufficiente e se nessuno ha capacità residua sufficiente se ne usa uno nuovo.

Si aggiunge quindi alla soluzione il nuovo assegnamento:

$$x^{(t+1)} = x^{(t)} \cup \{(i, j)\}$$

La soluzione così ottenuta **non è ottima**, ma è **approssimata**:

1. Occorrono almeno $f^* \geqslant \sum_{i \in O} \frac{v_i}{V}$ contenitori.
2. I contenitori usati, tranne al più l'ultimo, hanno contenuto maggiore di $\frac{V}{2}$: gli oggetti del secondo semi-vuoto sarebbero finiti nel primo.
3. Il volume totale supera quello degli $f_A - 1$ contenitori pieni.

$$\sum_{i \in O} v_i > (f_a - 1)\frac{V}{2}$$

4. Ne segue che: $f_A \leqslant 2f^* + 1$

Il fattore $\alpha = 2$ vale prendendo gli oggetti in qualsiasi ordine, ma è possibile migliorare l'approssimazione:

Se si prendono gli oggetti in ordine di volume **decrescente** il fattore migliora: $f_A \leqslant \frac{11}{9} f^* + 1$.

5.2.2 Set covering approssimato

Data una matrice binaria e un vettore di costi associati alle colonne, si cerca il sottoinsieme di colonne di costo minimo che copra tutte le righe.

Si procede a includere nel set $\mathrm{Ext}_A(x)$ solo le colonne che coprono righe aggiuntive e viene utilizzata come funzione di scelta:

$$\phi_A(i, x) = \frac{c_i}{a_i(x)}$$

dove $a_i(x)$ è il numero di righe coperte da i ma non da x.

Anche questo algoritmo costruttivo è approssimato, ma con approssimazione logaritmica:

$$f_A \leqslant (\ln|R| + 1)\, f^*$$

5.2.3 Nearest Neighbour per TSP

Si costruisce il set $\mathrm{Ext}_A(x)$ con gli archi uscenti dall'ultimo nodo del cammino x che non chiudono sotto-cicli.

$$\mathrm{Ext}_A(X) = \{(h, k) \in A : h = \mathrm{Last}(x), k \notin N_x \ \vee \ k = 1 \ \wedge \ N_x = N\}$$

dove N_x è l'insieme dei nodi visitati da x e $\mathrm{Last}(x)$ è l'ultimo nodo visitato.

Si parte con l'insieme degli archi vuoto (un cammino degenere che parte da una radice scelta) e si cerca l'arco di costo minimo uscente dall'ultimo nodo (allo step 0 è la radice). Questo procedimento si ripete sino a che non si crea un ciclo che torna alla radice.

NB: è il set $\mathrm{Ext}_A(x)$ che impone che la radice sia raggiunta per ultima.

5.2.4 Cheapest Insertion per TSP

Parte con un insieme di archi vuoto, rappresentante un ciclo degenere centrato sulla nota radice e procede ad espanderlo scegliendo l'arco $(s_i, s_{i+1}) \in x$ ed il nodo $k \notin N_x$ tali che $\left(c_{s_i,k} + c_{k,s_{i+1}} - c_{s_i,s_{i+1}}\right)$ sia minimo. Se il nuovo ciclo ottenuto tocca tutti i nodi l'algoritmo termina, altrimenti continua ad espandere la soluzione parziale.

Anche questo algoritmo non è esatto ma 2-approssimato sotto condizione di disuguaglianza triangolare.

Complessità computazionale 5.2.1 (Cheapest Insertion per il TSP). L'algoritmo esegue n passi: ad ogni passo valuta $t\,(n-t)$ coppie arco-nodo, ogni valutazione richiede tempo costante ed eventualmente aggiorna la mossa migliore. La complessità totale risultante è:

$$\Theta(n^3)$$

È possibile ridurla a $\Theta\left(n^2 \log n\right)$) conservando gli inserimenti possibili per ogni nodo esterno in un min-heap.

5.2.5 Nearest Insertion per TSP

Parte, come il cheapest insertion, con un insieme di archi vuoto che rappresenta un ciclo degenere centrato sulla radice, quindi procede con due criteri:

Criterio di selezione: Sceglie il nodo k più vicino al ciclo x:

$$k = \operatorname*{argmin}_{l \notin N_x} \left(\min_{s_i \in x} c_{s_i,l} \right)$$

Criterio di inserimento: Sceglie l'arco (s_i, s_{i+1}) che minimizza f:

$$(s_i, s_{i+1}) = \operatorname*{argmin}_{s_i \in x} \left(c_{s_i,k} + c_{k,s_{i+1}} - c_{s_i,s_{i+1}}\right)$$

L'algoritmo continua fino a che non viene realizzato un ciclo che tocca tutti i nodi.

Pure questo algoritmo non è esatto ma 2-approssimato sotto condizione di disuguaglianza triangolare.

Complessità computazionale 5.2.2 (Nearest Insertion per TSP). Ad ogni passo valuta:

1. $(n-t)$ nodi e trova il più vicino al ciclo.
2. t archi e trova il più conveniente da togliere.

La complessità totale è $\Theta\left(n^3\right)$. Si può ridurre a $\Theta\left(n^2\right)$ conservando per ogni nodo esterno il nodo interno più vicino.

5.2.6 Farthest Insertion per TSP

La scelta del nodo più vicino al ciclo é naturale, ma ingannevole, dato che tutti i nodi vanno raggiunti prima o poi: conviene servire al meglio i nodi più fastidiosi, cioè lontani.

Pure questo algoritmo parte con un insieme di archi vuoto che rappresenta un ciclo degenere centrato sulla radice, quindi procede con due criteri:

Criterio di selezione: sceglie il nodo k più lontano dal ciclo x:

$$k = \operatorname*{argmax}_{l \notin N_x} \left(\min_{s_i \in x} c_{s_i,l} \right)$$

Criterio di inserimento: sceglie l'arco (s_i, s_{i+1}) che minimizza:

$$(s_i, s_{i+1}) = \operatorname*{argmin}_{s_i \in x} \left(c_{s_i,k} + c_{k,s_{i+1}} - c_{s_i,s_{i+1}}\right)$$

L'algoritmo nuovamente si interrompe quando tocca tutti i nodi.

Si tratta di un algoritmo $\log n$-approssimato sotto ipotesi di disuguaglianza triangolare.

Complessità computazionale 5.2.3 (Farthest Insertion per TSP). A ogni passo valuta:

1. $(n-t)$ nodi e trova il più lontano dal ciclo.
2. t archi e trova il più conveniente da togliere.

La complessità totale è $\Theta(n^3)$, ma si può ridurre a $\Theta(n^2)$ conservando il nodo del ciclo più vicino ad ogni nodo esterno.

5.2.7 Distance Heuristic per Steiner Tree Problem

Dato un grafo non orientato G, con costi sui lati interi e un sottoinsieme di vertici speciali $U \subset V$, si cerca un albero di costo minimo che connetta i vertici speciali.

L'idea è di aggiungere un vertice speciale per volta e fermarsi quando tutti i vertici speciali sono connessi. Per poter mantenere la connessione, occorre aggiungere all'albero non un solo lato, ma un intero cammino. Inoltre trovare il cammino minimo da un vertice speciale all'albero x è facile.

Ogni volta si può determinare efficientemente l'insieme B^+ dei nuovi lati.

5.3 Algoritmi euristici distruttivi

Si tratta dell'approccio complementare, con cui si parte dall'intero insieme base B e si elimina un elemento per volta, che viene scelto in modo da non uscire dallo spazio di ricerca $\mathscr{F}_A$ ed ottimizzando un opportuno criterio.

Questa categoria di algoritmo euristico termina quando non c'è più modo di rimanere nello spazio di ricerca.

Tipicamente sono meno usate perché spesso richiedono un numero di passi superiore, hanno maggiori probabilità di commettere passi sbagliati e la valutazione dei criteri è tipicamente più costosa.

Combinare euristiche costruttive con euristiche distruttive può essere utile quando l'euristica costruttiva produce soluzioni ridondanti e la soluzione migliore generata dall'euristica costruttiva non è l'ultima.

6 Meta-euristiche costruttive

6.1 Multi-start

Si definiscono diversi criteri di scelta, quindi si applicano in sequenza gli algoritmi che ne derivano e si restituisce la soluzione migliore.

Spesso si usa quando $\phi_A(i,x)$ contiene parametri numerici μ.

6.2 Roll-out

Data un'euristica costruttiva base A:

1. Si parte con un sottoinsieme vuoto: $x^{(0)} = \emptyset$.
2. Ad ogni step:
   (a) Si estende la soluzione in ogni modo lecito:
   $$x^{(t-1)} \cup \{i\}, \forall \text{Ext}_A(x)$$
   (b) Ad ogni estensione si applica l'euristica base e si calcola il valore della soluzione $x_A\left(x^{(t-1)} \cup \{i\}\right)$ che fa da stima del risultato.
   (c) Si usa la stima come funzione di scelta $\phi_A(i,x)$:
   $$i^{(t)} = \underset{i \in \text{Ext}_A(x^{(t-1)})}{\text{argmin}} f\left(x_A\left(x^{(t-1)} \cup \{i\}\right)\right)$$
3. Si termina quando $\text{Ext}_A(x)$ è vuoto.

Si parla di euristica costruttiva **single-step look-ahead**.

Lo schema è parametrico nell'euristica base scelta.

6.2.1 Proprietà delle euristiche roll-out

Il risultato dell'euristica roll-out domina quello dell'euristica di base se opportune condizioni sono verificate.

È possibile ibridare euristiche roll-out e multi-start: date più euristiche costruttive di base $A^{[1]}, \ldots, A^{[l]}$, si parte dal sottoinsieme vuoto e ad ogni passo t:

1. Per ogni possibile estensione $i \in \text{Ext}_A\left(x^{(t-1)}\right)$ si esegue ogni algoritmo base $A^{[i]}$ a partire da $x^{(t-1)} \cup \{i\}$.
2. Si ottiene una stima $f_{A^{[l]}}\left(x^{(t-1)} \cup \{i\}\right)$
3. Si esegue la mossa che produce la stima migliore:
   $$i^{(t)} = \underset{l=1,\ldots,\mathscr{L}}{\text{argmin}} \min_{i \in \text{Ext}_A\left(x^{(t-1)}\right)} f\left(x_{A^{[l]}} \cup \{i\}\right)$$
4. Si termina quando $\text{Ext}_A(x)$ è vuoto.

L'euristica roll-out ibrida domina ogni euristica di base.

La complessità cresce fortemente rispetto all'euristica di base A ma resta polinomiale (al limite viene moltiplicata per $|B|^2$).

6.3 Adaptative Research Technique (ART)

Spesso nei primi passi un'euristica costruttive include elementi apparentemente buoni che portano a soluzioni finali molto cattive: la meta-euristica ART prova a **vietare elementi** allo scopo di impedire che elementi ingannevoli guidino il sotto-insieme x sul cammino sbagliato in $\mathscr{F}_A$.

Vietando elementi delle soluzioni già visitate si impedisce di ottenere soluzioni uguali o simili ad esse, proprietà detta di **diversificazione**, che consente di ripetere l'euristica quasi indefinitivamente e modulare la distanza delle nuove soluzioni dalla vecchie.

Iniziando da una euristica costruttiva di base A, si procede a creare una lista di elementi vietati, inizialmente vuota $(l_i) = -\infty, \forall\, i \in B$. Quindi, ad ogni iterazione:

1. Applica l'euristica base A evitando gli elementi vietati $(l - l_i \leqslant d)$ e si ottiene quindi una soluzione $x^{[l]}$.
2. Si decide con probabilità π se vietare o no ogni elemento $i \in x^{[l]}$.
3. Conserva per ogni elemento vietato l'iterazione l_i di inizio del divieto.
4. Rimuove i divieti più vecchi di d iterazioni, valore chiamato **expiration time**.
5. Conserva la miglior soluzione trovata e le corrispondenti l_i
6. Apporta eventuali modifiche ausiliarie alla lista dei divieti.

Al termine restituisce la migliore soluzione individuata.

6.3.1 Taratura dei parametri

Il metodo nasce con almeno 3 parametri:

1. Il numero totale delle iterazioni $\mathscr{L}$.
2. La durata d del divieto.
3. La probabilità π del divieto.

Valutare sperimentalmente le prestazioni con valori diversi ha alcuni svantaggi, come le lunghe fasi sperimentali causate dal numero combinatoriale di combinazioni e si può rischiare l'overfitting.

L'eccesso di parametri è pertanto un aspetto indesiderabile, che tipicamente rivela uno studio insufficiente del problema e dell'algoritmo.

6.3.2 Intensificazione

Definizione 6.3.1 (Intensificazione). L'intensificazione è il meccanismo opposto alla diversificazione, cioè la concentrazione della ricerca sui sottoinsiemi più promettenti.

6.4 Semigreedy e GRASP

Un'euristica costruttiva non esatta ha almeno un passo t in cui sceglie un elemento che non porta ad alcuna soluzione ottima: l'algoritmo semi-greedy si basa sull'idea che l'elemento che mantiene la via verso l'ottimo è fra i migliori per la funzione obbiettivo, anche se non rigorosamente il migliore, per cui se non è possibile correggere la funzione obbiettivo si procede a scegliere con una distribuzione di probabilità che favorisca i migliori.

Questo approccio ha proprietà importanti:

Proprietà 6.4.1. Se c'è un cammino da $\emptyset$ a x^*, l'algoritmo può raggiungere una soluzione ottima.

Proprietà 6.4.2. Rieseguendo l'euristica più volte, questa genera soluzioni diverse e la probabilità di raggiungere l'ottimo cresce via via.

6.4.1 Grafi di costruzione stocastici

Il **grafo di costruzione** consente una trattazione formale del procedimento:

1. I nodi sono i sottoinsiemi validi: $x \in \mathscr{F}_A$
2. Gli archi legano sottoinsiemi separati da una sola mossa.
3. L'arco $(x, x \cup \{i\})$ ha peso $\phi_A(i, x)$.

I cammini massimali raffresentano l'esecuzione di un'euristica costruttiva, che partono dal sottoinsieme vuoto e terminano in un sotto-insieme non ampliabile che spesso è una soluzione ammissibile.

Un'euristica costruttiva con passi casuali associa ad ogni arco $(x, x \cup \{i\})$ la probabilità $\pi_A(i, x)$ di estendere il sotto-insieme x con l'elemento i.

Proprietà 6.4.3. Un'euristica costruttiva con passi casuali ha probabilità non nulla di raggiungere l'ottimo se e solo se esiste almeno un cammino di probabilità non nulla da $\emptyset$ e x^*.

Proprietà 6.4.4. La probabilità di raggiungere l'ottimo tende a 1 per $l \to +\infty$.

Definizione 6.4.5 (Random Walk). La **Random Walk** è un'euristica costruttiva in cui le probabilità sugli archi uscenti da ogni nodo sono uniformi: se un cammino verso l'ottimo esiste, la random walk lo trova, per quanto in un tempo lunghissimo.

Le euristiche costruttive con passi casuali favoriscono gli archi più promettenti e sfavoriscono gli altri. C'è da tenere a mente però durante la modellazione che l'introduzione di archi a probabilità zero può sbarrare la via all'ottimo.

6.4.2 GRASP: Greedy Randomized Adaptive Search Procedure

Si tratta di una variante sofisticata dell'euristica semi-greedy. Può utilizzare diverse distribuzioni di probabilità discrete, tra cui:

Definizione 6.4.6 (Probabilità uniforme). Ogni arco uscente da x ha ugual probabilità di capitare in un altro stato. L'algoritmo, usando questa distribuzione, compie una random walk.

Definizione 6.4.7 (Heuristic-Biased Stochastic Sampling (HBSS)). Ordina gli archi uscenti da x per valori non decrescenti della funzione obbiettivo ed assegna probabilità decrescenti secondo la posizione nell'ordine, in base ad uno schema semplice: lineare, esponenziale, …

Definizione 6.4.8 (Restricted Candidate List (RCL)). Ordina gli archi uscenti da x per valori non crescenti della funzione obbiettivo ed inserisce i primi archi in una lista. Procede quindi ad assegnare probabilità uniforme ai primi archi, ed azzera la probabilità degli altri.

Tipicamente il numero degli archi è scelto o per **cardinalità** o per un **valore di threshold**.

La strategia più comunemente utilizzata è RCL.

Taratura reattiva dei parametri

Tipicamente si procede sfruttando la memoria: cioè si cerca di apprendere dai risultati precedenti.

1. Si fissa $\mathscr{L}$ e si scelgono m configurazioni di parametri $\mu_1, \ldots, \mu_m$
2. Si pone $l_r = \frac{l}{m}\ \forall r = 1, \ldots, m$
3. Si prova ogni configuzione μ_r per $\mathscr{L}_r$ iterazioni
4. Si valuta la media campionaria $\overline{f}(\mu_r)$ dei risultati ottenuti con μ_r.
5. So rimodula il numero di iterazioni $\mathscr{L}_r$ per ogni μ_r in base a $\overline{f}(\mu_r)$:

$$\mathscr{L}_r = \frac{{}^1/_{\overline{f}(\mu_r)}}{\sum_{s=1}^{m} {}^1/_{\overline{f}(\mu_s)}} \quad \forall r = 1, \ldots, m$$

   in modo che le configurazioni più efficaci facciano più iterazioni.

6. Si ripete l'intero procedimento, tornando al punto 3, per R volte.

6.5 Ant System (AS)

Definizione 6.5.1 (Stigmergia). Comunicazione indiretta fra agenti nella quale essi sono stimolati e guidati dai risultati delle azioni proprie e altrui.

Ogni agente é un'applicazione dell'euristica costruttiva base:

1. Lascia sui dati una traccia che dipende dalla soluzione prodotta.
2. Compie scelte influenzate dalle tracce lasciate dagli altri agenti

Le scelte dell'agente hanno inoltre anche una componente casuale.

È diversa dall'euristica semi-greedy:

1. Ogni iterazione lancia f volte l'euristica A.
2. Tutte le scelte di $\text{Ext}_A(x)$ sono considerate lecite, non vi è quindi una RCL.
3. La probabilità di una scelta deriva dal criterio $\phi_A(i,x)$ e da un'informazione ausiliaria chiamata **traccia** prodotto nelle iterazioni precedenti e talvolta dagli altri agenti nella stessa iterazione.

6.5.1 Traccia

La traccia è inizialmente uniforme e viene poi modulata per favorire le scelte promettenti e diminuendola per evitare le scelte troppo ripetitive.

6.5.2 La scelta casuale

Gli elementi sono estratti da $\text{Ext}_A(x)$ con probabilità:

$$\pi_A(i,x) = \frac{\eta_A(i,x)^{\mu_\eta}\,\tau_A(i,x)^{\mu_r}}{\sum_{j\in\text{Ext}_A(x)}\eta_A\left(j,x\right)^{\mu_\eta}\,\tau_A\left(j,x\right)^{\mu_r}}$$

Dove il denominatore normalizza la probabilità mentre la funzione ausiliaria $\eta_A(i,x)$ è detta **visibilità**:

$$\eta_A(i,x) = \begin{cases} \phi_A(i,x) & \text{per problemi di massimizzazione} \\ \frac{1}{\phi_A(i,x)} & \text{per problemi di minimizzazione} \end{cases}$$

Gli altri parametri, μ_τ e μ_η modulano i peso dei due termini, in particolare il peso dei dati e della memoria:

1. Per $\mu_\eta \approx 0 \wedge \mu_\tau \approx 0$ spingono verso la casualità (cioè diversificano).
2. Per $\mu_\eta >> \mu_\tau$ vengono favoriti i dati e si tende a seguire di più l'euristica costruttiva di base, intensificando quindi le soluzioni più immediate.
3. Per $\mu_\eta << \mu_\tau$ viene favorita la memoria e si tende a ritrovare soluzioni più vicine alle ultime trovate.

La variante detta **Ant Colony System** sceglie ad ogni passo:

1. Con probabilità q l'elemento che massimizza $\eta_A(i,x)\,\tau_A(i,x)^{\mu_\tau}$.
2. Con probabilità $1-q$ un elemento estratto uniformemente a caso.

Questo porta a comportamenti estremi simili a quelli sovra-descritti:

$q \approx 0$ Vengono fatte scelte casuali.

$q \approx 1 \wedge \mu_\tau \approx 0$ Vengono privilegiati i dati.

$q \approx 1 \wedge \mu_\tau$ **alto** Vengono privilegiate le scoperte fatte via via.

6.5.3 Aggiornamento della traccia

Ad ogni iterazione $l \in \{1, \ldots, \mathscr{L}\}$:

1. Si eseguono f istanze dell'euristica di base A.
2. Si gestisce un sotto-insieme di soluzioni $\overline{X}^{[l]}$ i cui elementi saranno favoriti nelle seguenti iterazioni.
3. Si aggiorna la traccia secondo la formula:

$$\tau_A(i,x) = \left(1-\rho\right)\tau_A(i,x) + \rho \sum_{y \in \overline{X}^{[l]} : i \in y} F_A\left(y\right)$$

Dove ρ è un parametro chiamato **oblio**: per valori vicini a 1 la vecchia traccia tende ad essere cancellata (che comporta *diversificazione*), mentre per valori prossimi a 0 viene conservata senza modifiche (che comporta *intensificazione*).

$F_A(y)$ invece è una **funzione di fitness** che esprime la qualità della soluzione y e garantisce $F > \tau$.

L'aggiornamento comporta due effetti: tende ad aumentare la traccia sugli elementi inclusi nelle soluzioni di $\overline{X}^{[l]}$ mentre diminuisce la traccia sugli elementi che non vi appartengono.

Teorema 6.5.2. Alcune varianti di Ant System convergono all'ottimo con probabilità 1 per $\mathscr{L} \to \infty$.

7

Algoritmi di ricerca locale

7.1 Gli algoritmi di scambio

In Ottimizzazione Combinatoria ogni soluzione x è un sotto-insieme di B. Un'**euristica di scambio** aggiorna passo per passo un sotto-insieme $x^{(t)}$:

1. Parte da una soluzione ammissibile $x^{(0)} \in X$ trovata in qualche modo.
2. Scambia coppie (A, D) di sotto-insiemi, A esterno e D interno a $x^{(t)}$, generando una famiglia di soluzioni ammissibili.

$$x'_{A,D} = x \cup A \setminus D \quad A \subseteq B \setminus X \wedge D \subseteq x$$

3. Ad ogni passo t, sceglie quali sotto-insiemi scambiare in base a un opportuno criterio di scelta $\phi(x, A, D)$.

$$\left(A^*, D^*\right) = \operatorname*{argmin}_{A,D} \phi(x, A, D)$$

4. Genera la nuova soluzione corrente eseguendo le modifiche scelte:

$$x^{(t+1)} = x^{(t)} \cup A^* \setminus D^*$$

5. Se vale una condizione di fine l'euristica termina, altrimenti ricomincia dal secondo step.

I due elementi fondamentali di un'euristica di scambio sono le coppie di sotto-insiemi disponibili per uno scambio a partire da x ed il criterio di scelta ϕ.

Definizione 7.1.1 (Intorno). $N : X \to 2^X$ è una funzione che associa a ogni soluzione ammissibile $x \in X$ un sotto-insieme di soluzioni ammissibile $N(x) \subseteq X$.

Una tipica definizione di intorno di x, con un parametro intero k, è l'insieme delle soluzioni con distanza di Hamming non superiore a k da x.

$$N_{H_k}(x) = \left\{x' \in X : d_H\left(x, x'\right) \leqslant k\right\}$$

Un'altra definizione comune è il numero di operazioni, quali **aggiunta**, **eliminazione** e **scambio**, che separano due soluzioni.

Definizione 7.1.2 (Grafo di ricerca). Un grafo i cui nodi rappresentano le soluzioni ammissibili $x \in X$ e gli archi collegano ogni soluzione x a quelle del suo intorno $N(x)$

Un arco viene detto **mossa** perché trasforma una soluzione in un'altra muovendo elementi.

Definizione 7.1.3 (Vettore di Incidenza). Una soluzione $x \in X$ può essere rappresentata col proprio vettore di incidenza:

$$x_i = \begin{cases} 1 & i \in x \\ 0 & i \in B \setminus x \end{cases}$$

Definizione 7.1.4 (Distanza di Hamming). La distanza di Hamming tra due soluzioni x ed x' è pari al numero di elementi per cui i loro vettori di incidenza differiscono:

$$d_H\left(x, x'\right) = \sum_{i \in B} \left|x_i - x'_i\right|$$

7.1.1 Connettività dello spazio delle soluzioni

Un'euristica di scambio può trovare soluzione ottima solo se almeno una soluzione otttima è raggiungibile da ogni soluzione iniziale.

Definizione 7.1.5 (Grado connesso all'ottimo). Si dice che il grafo di ricerca è **connesso all'ottimo** quando contiene per ogni soluzione $x \in X$ un cammino da x a X^*

Definizione 7.1.6 (Grafo fortemente connesso). Si dice che il grafo di ricerca è fortemente connesso quando contiene per ogni coppia di soluzioni $x, y \in X$ un cammino da x a y.

7.1.2 Euristiche steepest descent

Molto spesso il criterio di scelta ϕ della nuova soluzione in $N(x)$ è la funzione obiettivo, cioè ad ogni passo dell'euristica ci si sposta dalla soluzione corrente alla miglior soluzione del suo intorno. Per evitare comportamenti ciclici, si accettano solo soluzioni miglioranti.

Definizione 7.1.7 (Intorno Esatto). Un intorno esatto è una funzione intorno $N : X \to 2^X$ tale che ogni ottimo locale è anche ottimo locale:

$$\overline{X}_N = X^*$$

Sono estremamente rari. Un caso rilevante è lo scambio fra variabili di base e fuori base usato dall'**algoritmo del simplesso**.

In generale l'euristica steepest descent non trova l'ottimo globale.

7.1.3 Fitness-Distance correlation

Se la correlazione tra qualità e vicinanza agli ottimi globali è forte conviene costruire buone soluzioni iniziali perché avvicinano la ricerca locale a buoni ottimi locali, conviene quindi intensificare piuttosto che diversificare.

Al contrario, se la correlazione è debole, una buona inizializzazione è meno importante e conviene diversificare piuttosto che intensificare.

7.1.4 Landscape

Definizione 7.1.8 (Landscape). Si definisce **landscape** la terna (X, N, f) dove:

X è lo spazio di ricerca.

$N : X \to 2^X$ è la funzione intorno.

$f : X \to \mathbb{N}$ è la funzione obbiettivo.

Si può vedere come il grafo di ricerca persato sui nodi con l'obbiettivo.

Complessità di un landscape

La complessità di un landscape si può stimare empiricamente in vari modi:

1. Eseguendo una random walk nel grafo di ricerca.
2. Determinando la sequenza di valori dell'obbiettivo.
3. Calcolandone il valore medio campionario.
4. Calcolando il **coefficiente empirico di auto-correlazione**.

Definizione 7.1.9 (Coefficiente di autocorrelazione). Il coefficiente empirico di auto-correlazione:

$$r(i) = \frac{\frac{\sum_{t=1}^{t_{\max}-1}\left(f^{(t)}-\overline{f}\right)\left(f^{(t+i)}-\overline{f}\right)}{t_{\max}-i}}{\frac{\sum_{t=1}^{t_{\max}}\left(f^{(t)}-\overline{f}\right)^2}{t_{\max}}}$$

Si tratta di una funzione di i che parte da $r(0) = 1$, e in genere va calando.

Se rimane a circa 1 il landscape risulta **liscio**, se varia bruscamente il landscape è **accidentato**.

7.1.5 Plateau

Si può analizzare il grafo di ricerca dividendolo per livelli di obiettivo.

Definizione 7.1.10 (Plateau). Un plateau di valore f è ciascun sottoinsieme di soluzioni di valore f che siano adiacenti nel grafo di ricerca.

Plateau molto ampi ostacolano la scelta della soluzione dell'euristica steepest descent, perché fanno dipendere dall'ordine di visita di $N(x)$.

7.1.6 Bacini di attrazione

Definizione 7.1.11 (Bacini di attrazione). Un **bacino di attrazione** di una soluzione di ottimo locale $\overline{x}$ è l'insieme delle soluzioni $x^{(0)} \in X$ tali che l'euristica steepest descent partendo da $x^{(0)}$ produca come risultato $\overline{x}$.

L'euristica ha tipicamente risultati buoni se i bacini sono pochi e ampi.

7.1.7 Esplorazione dell'intorno

Due strategie sono possibili: la **ricerca esaustiva** in cui si valutano tutte le soluzioni dell'intorno, che spesso richiede molto tempo, o l'**esplorazione efficiente dell'intorno**, in cui anziché visitare tutto l'intorno si trova la sua soluzione ottima risolvendo qualche forma di problema ausiliario.

7.1.8 Conservazione parziale dell'intorno

Eseguendo un'operazione $o \in \mathcal{O}$ su due soluzioni simili $x, x' \in X$ spesso:

1. L'operazione è ammissibile per entrambe le soluzioni
2. La variazione della funzione obbiettivo è la stessa

In tali casi conviene:

1. Calcolare la variazione della funzione obbiettivo per ogni operazione ammissibile e conservare i valori a parte.
2. Eseguire l'operazione o^* migliore, generando la nuova soluzione x'.
3. Calcolare la variazione della funzione obbiettivo della nuova soluzione x' solo per le operazioni valide tranne quelle utilizzate per arrivare in questa soluzione e recuperare i valori salvati per $o \in \mathcal{O}_{xx'}$ che sono ancora validi.
4. tornare al punto 2.

8

Intorni di dimensione esponenziale

8.1 Very Large Scale Neighbourhood (VLSN) Search

Si tratta di approcci utilizzati per esplorare intorni esponenziali in $|B|$ che vengono visitati in tempo polinomiale: esistono due strategie principali, o si esplora euristicamente l'intorno, restituendo una soluzione candidata anziché la migliore dell'intorno stesso, o si sceglie un intorno nel quale l'obbiettivo possa essere ottimizzato in tempo polinomiale, anche se il numero di soluzioni è esponenziale.

8.2 Variable Depth Search (VDS)

Intorni basati su operazioni sono facilmente parametrizzabili, e in questi casi l'idea è di:

1. Definire una **mossa composta** come sequenza di mosse elementari.
2. Costruire la sequenza ottimizzando ogni passo elementare.
3. Accettare la soluzione finale solo se migliora quella iniziale.

8.2.1 Schema

Data una $x^{(t)}$, per ogni $x' \in N\left(x^{(t)}\right)$, anziché limitarsi a valutare $f\left(x'\right)$:

1. Si sceglie la miglior soluzione x'' in un intorno $\overline{N}\left(x'\right) \subseteq N\left(x'\right)$
2. Se x'' migliora rispetto alla soluzione iniziale x, esegue la mossa e torna al punto 1, altrimenti termina.
3. Restituisce quindi la miglior soluzione trovata durante l'intero procedimento.

8.2.2 Differenze rispetto alla steepest descent

1. Trova un ottimo locale per ogni soluzione dell'intorno come se eseguisse una specie di look-ahead a un passo.
2. Ammette peggioramenti lungo la sequenza di mosse elementari.
3. Deve evitare che le mosse elementari della sequenza si elidano a vicenda creando un ciclo.
4. Spesso usa strategie di modulazione fine per migliorare l'efficienza.

8.3 L'algoritmo di Lin-Kernighan per il TSP

L'algoritmo di Lin-Kernighan applica la VDS a sequenze di scambi 2-opt, cioè ogni scambio k-opt equivale a una sequenza di $(k-1)$ scambi 2-opt, ognuno dei quali cancella uno dei due archi aggiunti dal precedente.

8.3.1 Dettagli implementativi

Per evitare di distruggere le mosse già fatte, il secondo arco che viene cancellato ogni volta deve appartenere alla soluzione iniziale, e questo implica un limite superiore alla lunghezza della sequenza. Si dimostra che interrompere la sequenza appena gli scambi non migliorano più la soluzione iniziale non va a danneggiare il risultato.

La variazione complessiva dell'obbiettivo è la somma delle variazioni dovute ai singoli scambi. Ogni sequenza di numeri con somma negativa ammette una permutazione ciclica le cui somme parziali sono tutte negative.

Quindi, esiste una permutazione ciclica della sequenza di mosse che sia vantaggiosa ad ogni passo.

8.4 Metodi destroy-and-repair

Aggiunte ed eliminazioni combinate producono scambi, ma i sottoinsiemi ottenuti tramite scambi di elementi simili potrebbero essere inammissibili o di pessima qualità. Inoltre allargare l'intorno a scambi di più elementi può essere inefficiente ed in molti problemi la cardinalità delle soluzioni non è uniforme.

Un'alternativa è pertanto:

1. Cancellare un sotto-insieme $D \subset x$ di cardinalità $\leqslant k$ e completarla con un'euristica costruttiva.
2. Aggiungere un insieme $A \subset B \setminus x$ di cardinalità $\leqslant k$ e sfrondarla con un'euristica distruttiva.

8.5 Visita efficiente di intorni esponenziali

Un'altra famiglia di metodi si basa su intorni la cui soluzione ottima si possa trovare risolvendo un problema su un'opportuna matrice o grafo che vengono in genere definiti di miglioramento.

1. Packing: Dynasearch.
2. Ciclo di costo negativo: scambi ciclici.
3. Cammino minimo: ejection chains, order-and-split

8.5.1 Dynasearch

La variazione $\delta f^{(o)}(x)$ della funzione obbiettivo a seguito di una mossa elementare $o \in \mathcal{O}$ spesso dipende solo da una parte della soluzione.

Operazioni o' che agiscono su altre parti della soluzione hanno un effetto indipendente: non importa l'ordine con cui si eseguono. Se f risulta additiva, l'effetto dei due scambi semplicemente si somma.

Definizione 8.5.1 (Mossa composta). La mossa composta è un insieme di mosse elementari come nella VDS, però le mosse elementari devono avere effetti mutualmente indipendenti sull'ammissibilità e sull'obbiettivo.

La situazione si può modellare con una matrice di miglioramento A in cui:

1. Le righe rappresentano le componenti della soluzione.
2. Le colonne rappresentano le mosse elementari: ogni colonna ha un valore pari al miglioramento $-\delta f$ dell'obbiettivo.
3. Se la mossa j impatta sulla componente i, $a_{ij} = 1$, altrimenti $a_{ij} = 0$.

Si vuole determinare l'**impaccamento ottimo delle colonne**, cioè il sotto-insieme di colonne di valore massimo che non coprano la stessa riga.

Il *Set Packing Problem* è in genere *NP*-difficile, ma:

1. Su particolari matrici è polinomiale.
2. Se ogni mossa tocca al massimo due componenti allora:
   (a) Le righe diventano nodi
   (b) Le colonne diventano lati
   (c) Ogni impaccamento di colonne diventa un accoppiamento

   dunque un problema di accoppiamento massimo, che è polinomiale.

8.5.2 Scambi ciclici

In molti problemi le soluzioni ammissibili partizionano gli elementi in componenti a cui è associata l'ammissibilità e la funzione obbiettivo è addittiva rispetto alle componenti.

È naturale definire per questi problemi l'insieme di operazioni $\mathcal{T}_k$ che contiene i trasferimenti di k elementi dalla propria componente a un'altra.

Da $\mathcal{T}_k$ deriva il relativo intorno $N_{\mathcal{T}_k}$:

1. Spesso i vincoli di ammissibilità ostacolano i trasferimenti semplici.
2. Ma il numero dei trasferimenti multipli cresce rapidamente con k

Vogliamo trovare un sottoinsieme di $N_{\mathcal{T}_k}$ efficientemente esplorabile.

Definizione 8.5.2 (Grafo di miglioramento). Il grafo di miglioramento permette di descrivere sequenze di trasferimenti:

1. Un **nodo** i corrisponde a un elemento i dell'insieme base B.
2. Un **arco** (i, j) corrisponde al:
   (a) Trasferimento dell'elemento i dalla sua componente attuale S_i alla componente attuale S_j dell'elemento j.
   (b) Eliminazione dell'elemento j.
3. Il **costo dell'arco** c_{ij} corrisponde alla variazione della componente dell'obbiettivo associata a S_j.

$$c_{ij} = f\left(S_j \cup \{i\} \setminus \{j\}\right) - f\left(S_j\right)$$

   con $c_{ij} = +\infty$ se è inammissibile trasferire i eliminando j

Un ciclo di tale grafo corrisponde a una sequenza chiusa di trasferimenti ed il costo del ciclo corrisponde al costo della sequenza, ma solo se ogni nodo sat in una diversa componente.

Ricerca del ciclo di costo minimo

Il problema è *NP-difficile* ma:

1. Il vincolo di dover toccare una volta sola ogni componente permette algoritmi di programmazione dinamica abbastanza efficienti.
2. Una sequenza di numeri con somma negativa ammette sempre una permutazione ciclica con somme parziali tutte negative e quindi si possono scartare tutti i percorsi parziali di costo $\geqslant 0$.

Esistono inoltre algoritmi polinomiali per calcolare cicli negativi qualsiasi ((Floyd-Warshall) ed i cicli di costo medio minimo (costo totale diviso il numero degli archi).

Questi cicli, pur non rispettando il vincolo sulle componenti, forniscono:

1. Una stima per difetto che può dimostrare l'inesistenza di cicli negativi.
2. Un ciclo negativo che può rispettare il vincolo per fortuna oppure essere modificato fino ad ottenerne uno.

Infine, esistono algoritmi polinomiali euristici.

8.5.3 Ejection chains: catene di scambi non cicliche

È possibile creare catene di trasferimenti non cicliche, ma aperte, in modo che le cardinalità delle componenti possa variare. Per ottenere queste strutture basta aggiungere al **grafo di miglioramento**:

1. Un nodo sorgente.
2. Un nodo per ogni componente.
3. Archi del nodo sorgente ai nodi associati agli elementi.
4. Archi dai nodi associati agli elementi ai nodi associati alle componenti.

A questo punto si cerca il cammino di **costo minimo** che vada dal nodo sorgente al nodo componente, toccando un solo nodo componente e che non tocchi mai più di un nodo associato ad ogni componente.

8.6 Order-first split-second

Il metodo Order-first split-second per i problemi di partizione:

1. Costruisce una permutazione iniziale degli elementi.
2. Partiziona gli elementi in componenti in modo ottimo sotto il vincolo aggiuntivo che elementi della stessa componente siano consecutivi nella permutazione iniziale.

Ovviamente, la soluzione dipende dalla permutazione iniziale: si procede quindi a ripetere la soluzione per diverse permutazioni creando un metodo a due livelli:

1. Al livello superiore, si modificano la permutazione.
2. Al livello inferiore, si ottimizza la partizione data la permutazione.

Anche in questo caso si sfrutta un grafo ausiliario:

Data la permutazione $(s_1, \ldots, s_n)$ degli elementi dell'insieme base E:

1. Ogni nodo s_i corrisponde a un elemento s_i dell'insieme base E, più un nodo s_0 fittizio.
2. Ogni arco s_i, s_j con $i < j$ corrisponde a una componente potenziale $S_l = \left(s_{i+1}, \ldots, s_j\right)$ formata dagli elementi della permutazione:
   (a) da s_i escluso
   (b) a s_j compreso
3. Il costo c_{s_i,s_j} corrisponde al costo della componente $f(S_l)$.
4. L'arco non esiste se la componente è inammissibile.

Di conseguenza:

1. Ogni cammino da s_1 a s_n rappresenta una partizione di B.
2. Il costo del cammino coincide con il costo della partizione.
3. Il grafo è aciclico: trovare il cammino ottimo costa $\mathcal{O}(m)$ dove $m \leqslant \frac{n(n-1)}{2}$ è il numero degli archi.

9 Multi-start, ILS e VNS

9.1 Condizione di termine

Se la ricerca si ripete o prosegue anziché terminare in un ottimo locale, idealmente potrebbe avere durata anche infinita. In pratica, si usano condizioni di termine assolute:

1. Un dato **numero totale di ripetizioni della ricerca locale** oppure un dato numero totale di esplorazioni dell'intorno
2. Un dato **tempo totale di esecuzione**
3. Un dato **valore dell'obiettivo**
4. Un dato **miglioramento dell'obiettivo** rispetto alla soluzione iniziale oppure "relative".
5. Un dato numero di ripetizioni o esplorazioni dell'intorno dopo l'ultimo miglioramento del risultato f^*
6. Un dato **tempo di esecuzione dopo l'ultimo miglioramento**
7. Un dato **valore minimo del rapporto fra miglioramento dell'obiettivo e numero di esplorazioni o tempo di esecuzione**.

9.2 Modificare la soluzione iniziale

Si possono creare soluzioni iniziali diverse tra loro o **generandole casualmente** o applicando diverse **euristiche costruttive** o ancora modificando **soluzioni generate dall'algoritmo di scambio**.

9.3 Metodi multi-start

I metodi multi-start costituiscono l'approccio classico:

1. Si progettano più euristiche costruttive.
2. Ogni euristica costruttiva genera una soluzione iniziale.
3. Ogni soluzione iniziale viene migliorata dall'algoritmo di scambio.

Gli svantaggi sono:

Scarso controllo: le soluzioni generate tendono a somigliarsi.

Impossibilità di proseguire a oltranza: il numero di ripetizioni è fisso.

Sforzo di progetto elevato: bisogna inventare molti algoritmi diversi

Nessuna garanzia di convergenza nemmeno in tempo infinito

Per superare gli svantaggi, oggi si preferiscono meta-euristiche costruttive con memoria o passi casuali. GRASP e Ant System includono per definizione una procedura di scambio.

Se l'euristica di scambio e il meccanismo di reinizializzazione sono buoni, la soluzione iniziale ha influenza solo nelle fasi iniziali della ricerca.

9.4 Sfruttare le soluzioni precedenti

L'idea è sfruttare la memoria delle soluzioni già visitate:

1. Conservare delle soluzioni di riferimento, tipicamente l'ottimo locale migliore trovato sinora ed eventualmente altri.
2. Generare la nuova soluzione iniziale modificando quelle di riferimento.

9.5 Iterated Local Search (ILS)

Si tratta di un algoritmo composto da 4 elementi principali:

1. Un'euristica di scambio **steepest descent** che produce ottimi locali.
2. Una procedura di perturbazione che genera le soluzioni iniziali.
3. Una condizione di accettazione che indica se cambiare la soluzione di riferimento x.
4. Una condizione di terminazione.

L'idea è che:

1. L'euristica di scambio esplora rapidamente un bacino di attrazione, terminando in un ottimo locale.
2. La procedura di perturbazione passa a un altro bacino di attrazione.
3. La condizione di accettazione valuta se il nuovo ottimo locale è un punto di partenza promettente per la successiva perturbazione.

9.6 Procedura di perturbazione

Sia $\mathcal{O}$ l'insieme che definisce l'intorno $N_{\mathcal{O}}$ dell'euristica di scambio.

La **procedura di perturbazione** esegue un'operazione $o \in \mathcal{O}'$ scelta a caso e deve essere $\mathcal{O}' \not\subseteq \mathcal{O}$, altrimenti l'euristica di scambio riporterebbe la soluzione x' all'ottimo locale iniziale x.

Due tipiche definizioni di $\mathcal{O}'$ sono sequenze di $k > 1$ operazioni di $\mathcal{O}$ o operazioni concettualmente diverse.

La difficoltà principale della ILS è nel graduare la perturbazione:

- Troppo forte, trasforma la ricerca in un restart casuale.
- Troppo debole, riporta sempre la ricerca all'ottimo iniziale.

Idealmente si vuole poter entrare in ogni bacino e uscire da ogni bacino.

9.7 Criterio di accettazione

Il criterio di accettazione bilancia intensificazione e diversificazione:

- Accettare solo soluzioni miglioranti favorisce l'**intensificazione**:

$$\text{Accept}\left(x', x\right) = \left(f\left(x'\right) < f\left(x^*\right)\right)$$

  La soluzione di riferimento è sempre la migliore trovata $x = x^*$.

- Accettare qualsiasi soluzione favorisce la **diversificazione**:

$$\text{Accept}\left(x', x\right) = \text{true}$$

  La soluzione di riferimento è sempre l'ultimo ottimo trovato: $x = x'$.

Strategie intermedie si possono definire in base a $\delta f = f\left(x'\right) - f\left(x^*\right)$:

- Se $\delta f < 0$, si accetta x' sempre.
- Se $\delta f \geqslant 0$, si accetta x' con probabilità $\pi\left(\delta f\right)$, dove π è una funzione non crescente.

I casi più tipici sono:

Probabilità costante : $\pi\left(\delta f\right) = \overline{\pi} \in (0,1) \quad \forall \delta f \geqslant 0$.

Probabilità monotona decrescente $\pi(0) = 1$ e $\lim_{\delta f \to +\infty} \pi(\delta) = 0$

Si può usare anche la memoria, accettando x' più facilmente se molte iterazioni sono passate dall'ultimo miglioramento di x^*.

9.8 Variable Neighbourhood Search (VNS)

Le differenze principali tra ILS e VNS stanno nell'usare il criterio di accettazione stretto: $f(x') < f(x^*)$ ed un **meccanismo di perturbazione adattativo** anziché fisso.

Nella VNS vengono spesso utilizzate anche tecniche di modifica dell'intorno, spesso basato sulla gerarchia di intorni.

Definizione 9.8.1 (Gerarchia di intorni). Con gerarchi di intorni si intende una famiglia di intorni d'ampiezza crescente rispetto ad un paramtro k.

$$N_1 \subset N_2 \subset \ldots \subset N_k \subset \ldots \subset N_{k_{\max}}$$

Tipicamente si usano intorni basati su distanza di Hamming o basati su k operazioni. La soluzione iniziale viene estratta casualmente da uno di questi intorni.

9.8.1 Meccanismo adattivo di perturbazione

Si parla di **variable neighbourhood** perché l'intorno da cui si estrae la nuova soluzione iniziale varia in base ai risultati dell'euristica di scambio:

- Se trova soluzioni migliori, si usa l'intorno più piccolo, generando una soluzione iniziale molto vicina (intensificazione).
- Se non trova soluzioni migliori, si usa un intorno un po' più grande, generando una soluzione iniziale un po' più lontana (diversificazione).

Il metodo ha tre parametri:

1. $k_{\min}$ individua l'intorno più piccolo da cui si generano nuove soluzioni.
2. $k_{\max}$ individua l'intorno più grande da cui si generano nuove soluzioni.
3. δk è la variazione del parametro k fra due intorni consecutivi usati.

L'euristica di scambio adotta per efficienza l'intorno più piccolo possibile.

Taratura dei parametri

Tarare l'intorno più piccolo Il valore di $k_{\min}$ deve essere:

- abbastanza alto da far uscire dal bacino di attrazione corrente.
- non così alto da far saltare i bacini di attrazione adiacenti.

In genere si parte con $k_{\min} = 1$, e poi si modula sperimentalmente.

Tarare l'intorno più grande Il valore di $k_{\max}$ deve essere:

- abbastanza alto da raggiungere qualsiasi bacino di attrazione utile.
- non così alto da raggiungere regioni inutili dello spazio delle soluzioni.

In genere si parte col diametro dello spazio di ricerca per l'intorno base e poi si modula sperimentalmente. Alcuni esempi sono:

MDP $\min(m, n-m)$

MAX-SAT e TSP n

Tarare la variazione Il valore di δk deve essere:

- abbastanza alto da raggiungere $k_{\max}$ in tempi ragionevoli.
- non così alto da impedire a ogni valore di k da svolgere il suo ruolo.

In genere si parte con $\delta k = 1$ e poi si modula sperimentalmente

9.8.2 Skewed VNS

Un criterio di accettazione più raffinato è accettare x' quando:

$$f\left(x'\right) z f\left(x\right) + \alpha d_H\left(x', x\right)$$

dove x è l'ottimo di partenza, d_H è la distanza di Hamming e α è un opportuno parametro.

Si favorisce la diversificazione accettando soluzioni peggioranti se lontane:

1. Con $\alpha \approx 0$ si tende ad accettare solo soluzioni miglioranti.
2. Con $\alpha >> 0$ si tende ad accettare qualsiasi soluzione.

Ovviamente si possono anche adottare le strategie viste per il ILS.

10

Variable Neighbourhood Descent e Dynamic Local Search

10.1 Variable Neighbourhood Descent (VND)

La **Variable NeighbourHood Descent** sfrutta il fatto che gli ottimi locali sono relativi all'intorno scelto: cambiando intorno, in genere un ottimo locale non è più tale.

Definito un insieme di intorni $N_1, \ldots, N_{k_{\max}}$:

1. Si parte con $k = 1$
2. Si trova un ottimo locale $\overline{x}$ rispetto a N_k con un'euristica **steepest descent**
3. Si aggiorna k
4. Se $\overline{x}$ è un ottimo locale per tutti gli N_k si termina, altrimenti si torna al punto 2.

Vi è una stretta relazione fra algoritmi VND e VNS: le differenze fondamentali sono che nella VND:

1. Ad ogni passo la soluzione corrente è la migliore nota.
2. Gli intorno sono esplorati anziché usati per estrarre soluzioni casuali, quindi non sono mai enormi.
3. Gli intorni non formano necessariamente una gerarchia, quindi l'aggiornamento di k può non essere un incremento.
4. Quando si raggiunge un ottimo locale per ogni N_k, si termina.

10.1.1 Strategie di scorrimento degli intorni nella VND

Ci sono due categorie principali di metodi VND:

1. Nei metodi con **intorno gerarchico** si vuole:
   (a) Sfruttare a fondo gli intorni piccoli e rapidi.
   (b) Ricorrere a quelli grandi e lenti solo per uscire dagli ottimi locali

   Di conseguenza l'aggiornamento di k funziona come nella VNS:
   (a) quando non si trovano miglioramenti in N_k, si incrementa k.
   (b) quando si trovano miglioramenti in N_k, k torna a 1.
2. Nei metodi a **intorno eterogeneo** si vuole sfruttare la potenzialità di intorni topologicamente diversi tra loro, di conseguenza k scorre progressivamente i valori da 1 a $k_{\max}$.

Si termina quando la soluzione corrente è ottimo locale per tutti gli N_k:

1. Nel caso gerarchico, si termina quando fallisce $N_{k_{\max}}$.
2. Nel caso eterogeneo, occorre un segnalatore di miglioramento (flag).

10.2 Dynamic Local Search (DLS)

La **Dynamic Local Search** è un approccio complementare alla VND, conserva l'intorno iniziale e modifica la funzione obiettivo.

Si usa spesso nei problemi in cui l'obbiettivo è poco utile (per esempio nel caso di ampi *plateau*).

L'idea fondamentale è di:

- Associare all'insieme una funzione di penalità $w : B \to \mathbb{N}$.
- Costruire una funzione ausiliaria $\overline{f}\left(f(x), w(x)\right)$ che combina la funzione obbiettivo f con la penalità w.
- Applicare un'euristica di scambio **steepest descent** che ottimizza $\overline{f}$.
- Aggiornare la penalità w in base ai risultati e riapplicare l'euristica.

10.2.1 Varianti

L'idea possiede molte varianti:

1. Si possono applicare:
   (a) Penalità additive
   $$\overline{f}(x) = f(x) + \sum_{i \in x} w_i$$
   (b) Penalità moltiplicative
   $$\overline{f}(x) = \sum_{i \in x} w_i \phi_i$$
2. Le penalità possono subire:
   (a) Un aggiornamento casuale: perturbazione "rumorosa" dei costi
   (b) Un aggiornamento basato sulla memoria, che favorisca gli elementi più frequenti (intensificazione) o rari (diversificazione).
3. L'aggiornamento può avvenire:
   (a) Ad ogni singola esplorazione di intorno.
   (b) Quando si arriva a un ottimo locale per $\overline{f}$.
   (c) Quando la miglior soluzione nota x^* non cambia per parecchio.

10.2.2 DLS per il MCP

Dato un grafo non orientato, si cerca una clique di cardinalità massima:

- L'euristica di scambio è una VND che usa gli intorni.

  N_{A_1} **aggiunta di un vertice** La soluzione migliora sempre, ma l'intorno è molto piccolo e spesso vuoto.

  N_{S_1} **scambio di un vertice interno con un esterno** L'intorno è più grande, ma forma un *plateau*.
- L'obbiettivo non fornisce una direzione utile in nessuno dei due intorni.
- Si associa a ogni vertice i una penalità w_i, inizialmente nulla.
- L'euristica di scambio minimizza la penalità totale, all'interno dell'intorno.
- Si aggiorna la penalità:
  - Quando termina l'esplorazione di N_{S_1}: la penalità dei vertici della clique corrente aumenta di 1.
  - Dopo un dato numero di esplorazioni: tutte le penalità non nulle diminuiscono di 1.

La logica del metodo consiste nel tendere a:

- espellere i vertici interni.
- in particolare, quelli rimasti interni più a lungo.

10.2.3 DLS per il MAX-SAT

Date m disgiunzioni logiche dipendenti da n variabili logiche, si cerca un assegnamento di verità che soddisfi il massim numero di formule.

- L'intorno N_{F_1} è generato invertendo il valore di una variabile.
- Si associa a ogni formula logica una penalità w_j inizialmente pari a 1.
- L'euristica di scambio massimizza il numero di formule soddisfatte.
- La funzione modificata massimizza il numero più la penalità totale.
- Si aggiorna la penalità:
  - Quando si raggiunge un ottimo locale:

    $$w_j = \alpha_{\text{us}} w_j \quad \forall j \in U(x), \text{ le formule insoddisfatte}$$

    con α_{us} per favorire le formule attualmente insoddisfatte.
  - Con una certa probabilità o dopo un certo numero di aggiornamenti:

    $$w_j = (1-\rho) w_j + \rho \quad \forall j$$

    per riuniformare le penalità al valore minimo unitario.

La logica del metodo consiste nel tendere a:

- Soddisfare le formule attualmente insoddisfatte (diversificazione).
- In particolare, quelle rimaste insoddisfatte più a lungo e più di recente (memoria).
- Valori bassi di ρ, il parametro dell'oblio, conservano le penalità (intensificazione), valori alti le cancellano (diversificazione).

La taratura dei parametri può essere reattiva:

- Applicare per lo stesso tempo diverse configurazioni.
- Replicare il procedimento dando più tempo alle configurazioni sperimentalmente migliori.

11 Simulated Annealing e Tabù Search

Anziché ripetere la ricerca locale è possibile proseguirla, ma se non cambiano intorno e obbiettivo, bisogna accettare soluzioni non minime:

$$x' = \operatorname*{argmin}_{x \in N(x)} f(x)$$

ed eventualmente anche non miglioranti.

Il problema principale è il rischio di visitare ciclicamente soluzioni uguali.

Esistono principalmente due strategie che consentono di controllarlo:

1. Simulated Annealing, che usa passi causali.
2. Tabu Search, che usa memoria.

11.1 Simulated Annealing

Deriva dall'algoritmo di Metropolis, che intende simulare il processo di ricottura dei metalli, e possiede varie analogie con l'Ottimizzazione Combinatoria: questo suggerisce la possibilità di usarlo per l'ottimizzazione.

L'algoritmo di Metropolis genera una sequenza casuale di stati:

1. Lo stato corrente i ha energia E_i.
2. L'algoritmo perturba i, generando uno stato j con energia E_j.
3. Lo stato corrente passa da i a j con probabilità:

$$\pi_T(i,j) = \begin{cases} 1 & E_j < E_i \\ e^{\frac{E_i - E_j}{kT}} & E_j \geqslant E_i \end{cases}$$

Vale a dire che lo spostamento è deterministico se migliora, basato sulla probabilità condizionata se peggiora.

11.1.1 Criterio di accettazione

$$\pi_T(x,x') = \begin{cases} 1 & f(x') < f(x) \\ e^{\frac{f(x) - f(x')}{T}} & f(x') \geqslant f(x) \end{cases}$$

La temperatura T regola la probabilità di accettare peggioramenti:

1. Con $T >> 0$ vengono accettati quasi sempre: si tende a diversificare al limite come in una random walk.
2. Con $T \approx 0$ vengono rifiutati quasi sempre: si tende ad intensificare, al limite come in una steepest descent.

11.1.2 Convergenza all'ottimo

La probabilità che la soluzione corrente sia x' è la somma su tutti i possibili x delle probabilità di:

- Estrarre la mossa (x,x'), che è uniforme.
- Accettare la mossa è:

$$\pi_T(x,x') = \begin{cases} 1 & f(x') < f(x) \\ e^{\frac{f(x) - f(x')}{T}} & f(x') \geqslant f(x) \end{cases}$$

dunque, ad ogni passo dipende solo dalla probabilità al passo prima: la variabile aleatoria x forma nel tempo una **catena di Markov**.

Ad ogni temperatura fissata, le probabilità di transizione sono uniformi: si ha una catena di Markov omogenea.

Se lo spazio di ricerca è connesso rispetto all'intorno N, la probabilità di raggiungere ogni stato è > 0 e si ha una catena di Markov irriducibile.

Sotto queste ipotesi, la probabilità tende a una distribuzione stazionaria indipendente dalla soluzione iniziale.

La distribuzione stazionaria è quella indicata dalla termodinamica per l'equilibrio termico dei sistemi fisici che privilegia le soluzioni "buone", con X insieme delle soluzioni ammissibili e T parametro "temperatura":

$$\pi_T(x) = \frac{e^{\frac{-f(x)}{T}}}{\sum_{x \in X} e^{\frac{-f(x)}{T}}} \quad \forall x \in X$$

Se $T \to 0$ la distribuzione tende a una distribuzione limite:

$$\pi(x) = \lim_{T \to 0} \pi_T(x) = \begin{cases} \frac{1}{|X^*|} & x \in X^* \\ 0 & x \in X \setminus X^* \end{cases}$$

che corrisponde a una convergenza certa a una soluzione ottima globale.

Il risultato però vale all'equilibrio, e valori bassi di T implicano un'alta probabilità di visitare un ottimo globale e una convergenza lenta all'ottimo.

In un tempo finito, non sempre usare T più basso migliora il risultato.

D'altra parte, non è necessario visitare spesso le soluzioni ottime: basta aver visitato almeno una volta una soluzione ottima. In pratica, la temperatura T viene aggiornata: parte alta e va calando.

La temperatura T iniziale viene scelta abbastanza alta da consentire di accettare molte mosse ed abbastanza basso da rifiutare le mosse peggiori: un metodo classico è campionare l'intorno iniziale e fissare una temperatura tale da accettare una data frazione delle mosse nell'intorno (per esempio il 90%).

11.1.3 Aggiornamento della temperatura

Si procede per fasi successive ($r = 0, \ldots, m$):

1. Ogni fase applica un valore $T^{[r]}$ costante per $\mathscr{L}^{[r]}$ iterazioni.
2. $T^{[r]}$ viene via via aggiornata con un profilo esponenziale

$$T^{[r]} = \alpha^r T^{[0]}$$

3. $\mathscr{L}^{[r]}$ viene aggiornato
   - crescendo da una fase all'altra, spesso linearmente.
   - con valori legati al diametro del grafo di ricerca, quindi alla dimensione dell'istanza.

Siccome T è variabile, la catena di Markov x non è omogenea ma se T cala abbastanza lentamente converge all'ottimo globale ed i parametri per definire il calo dipendono dall'istanza, in particolare da $f\left(\overline{x}\right) - f\left(x^*\right)$, dove $\overline{x}$ è il miglior ottimo locale non globale.

11.1.4 Efficienza computazionale e varianti

Calcolare la probabilità con un'esponenziale può essere pesante: conviene precalcolare una tavola dei valori di $e^{\frac{\delta f}{T}} \quad \forall \delta f = f(x) - f\left(f'\right)$.

Se la temperatura T dipende dai risultati ottenuti, si para di SA adattativo. T è tarato in modo che una data frazione di $N(x)\ sialecita$, e cresce se la soluzione non migliora da molto, altrimenti cala.

11.2 Tabu Search

L'idea è vietare le soluzione già visitate, imponendo alla ricerca un tabu: il punto fondamentale è come rendere efficiente l'imposizione del divieto.

11.2.1 Ricerca locale con tabù

Un'euristica di scambio basata sull'esplorazione esaustiva dell'intorno con un tabù sulle soluzioni visitate richiede i seguenti passi:

1. Valutare l'ammissibilità di ogni sotto-insieme prodotto dagli scambi.
2. Valutare il costo di ogni soluzione ammissibile.
3. Valutare la condizione tabù di ogni soluzione ammissibile promettente.
4. Scegliere la miglior soluzione ammissibile non tabù.

Un modo elementare per realizzare la valutazione del tabù è salvare le soluzioni visitate in una struttura opportuna e confrontare ogni soluzione esplorata con quelle tabù.

11.2.2 Imporre il tabù in modo efficiente

Questa valutazione elementare del tabù però è molto inefficiente:

- Il confronto delle soluzioni al passo t richiede tempo $O(t)$.
- Il numero di soluzioni visitate cresce indefinitamente nel tempo.
- La memoria occupata cresce indefinitamente nel tempo

Il *Cancellation Sequence Method* ed il *Reverse Elimination Method* affrtontano questi problemi, sfruttando il fatto che in generale:

- Le soluzioni visitate formano una catena con piccole variazioni.
- Poche soluzioni visitate cadono nell'intorno di quella corrente.

L'idea è concentrarsi sulle variazioni.

Vietare le soluzioni visitate può avere due effetti negativi diversi:

- Può sconnettere il grafo di ricerca, creando delle "cortine di ferro" invalicabili che bloccano la ricerca: quindi sarebbe meglio evitare divieti assoluti.
- Può rallentare l'uscita dai bacini di attrazione, creando un effetto di riempimento graduale, che rallenta la ricerca: quindi sarebbe meglio allargare il divieto ad altre soluzioni.

I due fenomeni suggeriscono rimedi opposti.

11.2.3 Tabù basati su attributi

Consiste nel vietare le soluzioni con "attributi" comuni con le soluzioni visitate, anziché limitarsi a vietare le soluzioni visitate.

Come si procede:

- Si definisce un insieme A di attributi.
- Si gestisce un sotto-insieme $\overline{A}$ di attributi vietati.
- Sia $A_y \subseteq A$ il sottoinsieme di attributi posseduto dalla soluzione $y \in X$.
- Si vietano tutte le soluzioni dotate di attributi vietati:

$$A_y \cap \overline{A} \neq \emptyset \iff y \text{ è tabù}$$

- Se si esegue una mossa che trasforma la soluzione corrente da x a y si aggiungono ad $\overline{A}$ gli attributi che x aveva e y non ha:

$$\overline{A} = \overline{A} \cup \left(A_x \setminus A_y\right)$$

Questo significa che si evitano soluzioni simili a quelle già visitate e ci si allontana più in fretta dagli ottimi locali visitati.

11.2.4 Tabù temporanei e criteri di aspirazione

Siccome il tabù crea zone difficili o impossibili da raggiungere, è possibile fissare una **durata limitata** L, detta **tabù tenure**, con cui le soluzioni vietate tornano accessibili dopo un po' e si possono rivisitare le stesse soluzioni.

Siccome il tabù potrebbe vietare ottimi globali per semplice somiglianza si introduce un **criterio di aspirazione**: una soluzione tabù che sia migliore della miglior soluzione nota viene comunque accettata.

Nel caso estremo in cui tutte le soluzioni dell'intorno sono tanù, si accetta quella con tabù più vecchio.

11.2.5 Attributi tabù

Il concetto di attributo è volutamente generico, alcuni esempi sono:

Appartenenza di un elemento soluzione

Quando la mossa da x a y fa uscire un elemento i dalla soluzione, il tabù proibisce il reinserimento di i in soluzione.

Non appartenenza di un elemento alla soluzione

Quando la mossa da x a y fa entrare un elemento i dalla soluzione, il tabù proibisce l'eliminazione di i dalla soluzione.

Spesso vengono utilizzati più attributi insieme, ognuno con la sua tenure e lista.

Valore della funzione obbiettivo

Si vietano soluzioni di un dato valore già assunto in precedenza dall'obbiettivo.

Valore di una funzione ausiliaria

Per esempio la distanza dalla miglior soluzione nota.

11.2.6 Valutazione efficiente del tabù

Si può valutare il tabù in tempo costante con una struttura che associa ad ogni attributo l'iterazione di inizio del tabù.

Per vietare gli inserimenti ($A = x$), ad ogni iterazione t:

- valutando le mosse, è tabù inserire $i \in B \setminus x \quad \forall t \leqslant T_i^{\text{in}} + L^{\text{in}}$.
- eseguita la mossa, si pone $T_i^{\text{in}} = t \quad \forall i$ eliminato da x

Per vietare le eliminazioni ($A = B \setminus x$):

- valutando le mosse, è tabù eliminare $i \in x \quad \forall t \leqslant T_i^{\text{out}} + L^{\text{out}}$
- eseguita la mossa, si pone $T_i^{\text{out}} = t \quad \forall i$ inserito in x

Basta un vettore solo per vietare entrambe, dato che o $i \in x$ o $i \in B \setminus x$

11.2.7 Taratura della tabu tenure

Il valore più efficace di L è tipicamente legato alla dimensione della dimensione dell'istanza, spesso cresce lentamente (per esempio con la radicedi n) e valori quasi costanti funzionano bene su ampi intervalli di dimensione.

Estrarre L a caso da un intervallo $[L_{\min}, L_{\max}]$ rompe gli andamenti ciclici.

Le **tabu tenure adattive** reagiscono ai risultati della ricerca aggiornando L entro un prefissato intervallo $[L_{\min}, L_{\max}]$:

L **diminuisce quando la soluzione corrente** x **migliora:** si pensa di avvicinarsi a un ottimo locale nuovo e si vuole favorire la ricerca (intensificazione).

L **aumenta quando la soluzione corrente** x **peggiora** si pensa di allontanarsi da un ottimo locale visitato e non si vuole rallentare (diversificazione).

11.2.8 Varianti

Sul lungo periodo, anche i metodi adattati perdono di efficacia. Si adottano allora strategie di lungo termine:

Tabu Search reattivo: usa hash table per conservare le soluzioni visitate.

Frequency-based Tabu Search: conserva la frequenza di ogni attributo in soluzione in strutture analoghe a quelle usate per la recentezza.

Exploring Tabu Search: reinizializza la ricerca da soluzioni di buona qualità esplorate, ma mai assunte come soluzione corrente.

Granular Tabu Search: modifica l'intorno allargandolo via via.

12

Euristiche di ricombinazione

Le euristiche di ricombinazione, a differenza delle euristiche costruttive e di scambio (almeno tipicamente), gestiscono molte soluzioni in parallelo.

Esse partono da un insieme, detto **popolazione**, di soluzioni, detti **individui**, e ricombinano questi individui producendo una nuova popolazione.

Il loro aspetto originale è l'uso di operazioni che lavorano su più soluzioni.

12.1 Schema generale

L'idea di fondo è che:

1. Soluzioni buone condividono componenti con l'ottimo globale.
2. Soluzioni diverse possono condividere componenti diverse.
3. Combinando soluzioni diverse è possibile fondere componenti ottime più facilmente che costruendole un passo per volta.

Tipicamente si procede nelle modalità seguenti:

1. Costruire una popolazione iniziale di soluzioni.
2. Finché non si verifica un'opportuna condizione di termine produrre popolazioni successive, dette **generazioni**.
3. Per ogni generazione:
   (a) Estrarre sotto-insiemi di individui.
   (b) Applicare operazioni di scambio agli individui singoli.
   (c) Applicare operazioni di ricombinazione ai sotto-insiemi.
   (d) Raccogliere gli individui generati dalle operazioni.
   (e) Scegliere se accettare o no ogni nuovo individuo (e in quante copie) producendo così una nuova popolazione.

12.2 Scatter Search

Questa euristica di ricombinazione:

1. Genera un popolazione iniziale di scambio.
2. Le migliora con una procedura di scambio.
3. Costruisce un insieme di riferimento (reference set ($R = R_B \cup R_D$)).
   - Il sottoinsieme R_B contiene le migliori soluzioni note.
   - Il sottoinsieme R_D contiene le soluzioni più distanti tra loro e da R_B.
4. Per ogni coppia di soluzioni $x, y \in R_B \times (R_B \cup R_D)$:
   - Si procede a ricombinare x e y, generando z.
   - Migliora z ottenenendo z' con una procedura di scambio
   - Se $z' \notin R_B$ e in R_B c'è una soluzione peggiore, la sostituisce con z'.
   - Se $z' \notin R_D$ e in R_D c'è una soluzione più vicina, la sostituisce con z'.
5. La procedura termina quando R rimane invariato.

La logica è che le ricombinazioni $R_B \times R_B$ **intensificano** la ricerca, mentre le ricombinazioni $R_B \times R_D$ **diversificano** la ricerca.

12.3 Procedura di ricombinazione

La procedura di ricombinazione spesso dipende dalla natura del problema. Spesso ci si limita a trattare x e y come sotto-insiemi:

1. Prima z include tutti gli elementi condivisi da x e y: $z = x \cap y$.
2. Poi si procede ad aggiungere a z elementi estratti a caso alternativamente da $x \setminus y$ e da $y \setminus x$ fino a ottenere una soluzione ammissibile.

Se il sottoinsieme z ottenuto è inammissibile, un'euristica di scambio ausiliaria, detta **procedura di riparazione**, riporta z all'ammissibilità.

12.4 Path relinking

Viene usato generalmente come procedura finale di intensificazione più che come metodo a sé. Dato un intorno N su cui si basa un'euristica ausiliaria di scambio:

1. Raccoglie in un insieme di riferimento R le migliori soluzioni generate dall'euristica ausiliaria, dette **soluzioni di élite**.
2. Per ogni coppia di soluzioni $x, y \in R$:
   - Costruisce un cammino da x a y nello spazio di ricerca dell'intorno N applicando a $z^{(0)} = x$ l'euristica ausiliaria di scambio, ma scegliendo a ogni passo la soluzione più vicina alla destinazione y:

     $$z^{z+1} = \operatorname*{argmin}_{z \in N(z^{(k)})} D(z, y)$$

     dove R_D è un'opportuna funzione metrica sulle soluzioni. A pari distanza, ottimizza la funzione obbiettivo f del problema.
   - Trova la miglior soluzione z^*_{xy} lungo il cammino:

     $$z^*_{xy} = \operatorname*{argmin}_k f\left(z^{(k)}\right)$$

   - Se $z^*_{xy} \notin R$ ed è migliore di una di quelle di R, la inserisce in R.

12.4.1 Cammini di Relinking

I cammini esplorati in questo modo:

- Intensificano la ricerca, perché collegano soluzioni buone.
- Diversificano la ricerca, perché in genere sono diversi da quelli seguiti dall'euristica di scambio, soprattutto se gli estremi sono lontani.
- Poiché la distanza di $z^{(k)}$ da y cala via via, si possono esplorare:
  - Soluzioni peggioranti senza il rischio di comportamenti ciclici.
  - Sotto-insiemi inammissibili senza il rischio di non riuscire a riottenere soluzioni ammissibili.

12.4.2 Varianti

Esistono diverse varianti al Path Relinking, tra cui:

Backward path relinking costruisce il cammino all'indietro da y a x.

Back-and-forward path relinking costruisce entrambi i cammini.

Mixed path relinking costruisce un cammino facendo un passo per volta alternativamente da ogni estremo (aggiornando la destinazione).

Truncated path relinking costruisce solo il principo del cammino.

13

Algoritmi genetici ed evoluzionistici

13.1 Algoritmi operanti su codifiche

Molte euristiche di ricombinazione lavorano su **codifiche** delle soluzioni, dette **rappresentazioni compatte**, anziché sulle soluzioni stesse. Gli operatori di scambio e ricombinazione sono definiti sulle codifiche.

Le ragioni di questo approccio sono :

Astrazione Distinguere concettualmente il metodo dal problema cui viene applicato.

Generalità Costruire operatori di scambio e ricombinazione validi per ogni problema con un dato insieme di codifiche.

13.2 Algoritmo genetico

Questo approccio prevede la costruzione di una popolazione $X^{(0)}$ seguita ripetutamente da:

1. **Selezione:** genera una nuova popolazione a partire da quella corrente.
2. **Cross-over:** ricombina sotto-insiemi di due o più individui.
3. **Mutazione:** modifica singoli individui.

13.2.1 Caratteristiche di una buona codifica

Le prestazioni di un algoritmo genetico dipendono dalla codifica scelta, e buone codifiche dovrebbero avere le seguenti proprietà, con importanza decrescente:

1. Ogni soluzione deve avere una codifica, altrimenti vi sarebbero **soluzioni non ottenibili**.
2. Ogni codifica deve essere traducibile in una soluzione, altrimenti la popolazione conterrebbe **individui inutili**.
3. Ogni soluzione dovrebbe corrispondere a un pari numero di codifiche, altrimenti vi sarebbero **soluzioni indebitamente avvantaggiate**.
4. Le operazioni di codifica e decodifica dovrebbero essere efficienti, altrimenti si avrebbe un **algoritmo altamente inefficiente**.
5. Località: modifiche piccole alla codifica dovrebbero produrre modifiche piccole nella decodifica, altrimenti **non può intensificare**.

Queste condizioni dipendono moltissimo dai vincoli del problema.

In particolare, operatori mutazione e cross-over generici producono facilmente sottoinsiemi non ammissibili.

Definizione 13.2.1 (Vettore di incidenza). La codifica più diretta per problemi di Ottimizzazione Combinatoria è il **vettore binario di incidenza** $\xi \in \mathbb{B}^{|B|}$:

$$\begin{cases} \xi_i = 1 & i \in x \\ \xi_i = 0 & i \notin x \end{cases}$$

13.2.2 Codifica in stringhe di simboli

Se l'insieme base è partizionabile in componenti:

$$B = \bigcup_{c \in C} B_c \quad \text{con } B_c \cap B_{c'} = \emptyset \quad \forall c \neq c'$$

in modo che le soluzioni contengano un elemento di ogni componente:

$$|x \cap B_c| = 1 \quad \forall c$$

1. Definire per ogni componente c un alfabeto di simboli che descrive B_c.
2. Codificare la soluzione con una stringa di simboli $\xi \in B_1 \times \dots B_{|C|}$:

   $$\xi_c = \alpha \text{ indica che } x \cap B_c = (c, \alpha)$$

13.2.3 Permutazioni

Una codifica di uso comune è data dalle **permutazioni di un insieme**:

- Le soluzioni sono permutazioni, è la codifica naturale (per esempio nel TSP i nodi).
- Se le soluzioni sono partizioni e l'obbiettivo è additivo sui sottoinsiemi, il metodo *order-first split-second* trasforma permutazioni in partizioni.
- Se il problema ha un algoritmo costruttivo che ad ogni passo esegue:
  - Scelta di un elemento.
  - Scelta del modo di inserirlo in soluzione.

  È possibile scegliere gli elementi nell'ordine dato dalla permutazione.

13.2.4 Selezione

Ad ogni generazione g si costruisce una nuova popolazione $X^{\{g\}}$:

$$X^{(g)} = \text{Selezione}\left(X^{g-1}\right)$$

estraendo $n_p = \left|X^{(g)}\right|$ individui dalla popolazione corrente $X^{(g-1)}$.

L'estrazione segue due criteri fondamentali:

1. È lecito estrarre più volte lo stesso individuo.
2. La probabilità di estrazione è più alta per gli individui migliori.

   $$\phi(\xi) > \phi\left(\xi'\right) \Longrightarrow \pi_\xi \geqslant \pi_{\xi'}$$

   Dove la **fitness** ϕ è una misura di qualità dell'individuo ξ, legata al valore dell'obbiettivo per la corrispondente soluzione.

Esistono diversi schemi di selezione.

13.2.5 Selezione proporzionale

Semplicemente assegna una probabilità proporzionale alla fitness:

$$\pi_\xi = \frac{\phi(\xi)}{\sum_{\xi \in X^{(g-1)}} \phi(\xi)}$$

Definizione 13.2.2 (Roulette-wheel selection). Si parla di **roulette-wheel selection** o **spinning wheel selection**:

1. Data la popolazione $X^{(g)=\left\{\xi_1,\dots,\xi_{n_p}\right\}}$
2. Si costruiscono gli intervalli $\Gamma_i = \left(\sum_{k=1}^{i-1} \pi_{\xi_i}; \sum_{k=1}^{1} \pi_{\xi_i}\right]$
3. Si estrae un numero casuale $r \in U(;]$
4. L'individuo i^* scelto è quelle tale che $r \in \Gamma_{i^*}$

13.2.6 Selezione per rango

La selezione proporzionale soffre di:

Convergenza prematura: se i migliori individui sono cattivi e gli altri sono pessimi, la selezione produce rapidamente una popolazione cattiva.

Stagnazione: a lungo termine, tutti gli individui tendono ad avere una buona fitness, dunque uguale probabilità di selezione.

Per ovviare a questi difetti occorre allo stesso tempo sia limitare la differenza di probabilità tra individui diversi che differenziare la probabilità tra individui simili.

La selezione per rango ordina gli individui per fitness non decrescente ed assegna all'individuo k-esimo una probabilità:

$$\pi_{\xi_k} = \frac{2k}{n(n-1)}$$

Risulta però computazionalmente più gravoso che usare direttamente la fitness.

13.2.7 Selezione per torneo

È un compromesso che non richiede l'ordinamento completo delle fitness:

- Estrae n_p sotto-insiemi $\overline{X}_1, \ldots, \overline{X}_{n_p}$ di dimensione α
- Seleziona in ciascuno l'individuo di fitness massima:

$$\xi_k = \underset{\xi \in \overline{X}_k}{\operatorname{argmax}} \phi(\xi)$$

Il parametro α modula la forza della selezione:

$\alpha \approx n_p$ favorisce gli individui migliori.

$\alpha \approx 2$ lascia buone probabilità anche agli individui cattivi.

Tutte le procedure ammettono la variante elitista, che include nella nuova popolazione l'individuo migliore di quella corrente.

13.2.8 Crossover

Definizione 13.2.3 (Crossover). L'operatore di crossover combina due individui procedendone altri due.

Definizione 13.2.4 (Crossover semplice). Crossover con un singolo split.

1. Si estrae una posizione a caso con probabilità uniforme.
2. Si spezza la codifica in due parti in corrispondenza a tale posizione
3. Si scambiano le parti finali delle codifiche dei due individui.

Definizione 13.2.5 (Crossover doppio). Crossover con un doppio split.

1. Si estraggono due posizioni a caso con probabilità uniforme.
2. Si spezza la codifica in tre parti in corrispondenza a tali posizioni.
3. Si scambiano le parti estreme delle codifiche dei due individui.

Definizione 13.2.6 (Crossover ad α punti). Si tratta di una versione generalizzata dei precedenti:

1. Si estraggono α posizioni a caso con probabilità uniforme.
2. Si spessa la codifica in $\alpha + 1$ parti in corrispondenza a tale posizione.
3. Si scambiano le parti dispari delle codifiche dei due individui.

Per valori di α bassi si ha una **polarizzazione posizionale**, o **positional bias**: simboli vicini nella codifica tendono a rimanere insieme.

Definizione 13.2.7 (Crossover uniforme). È una versione di crossover che evita completamente il problema della **polarizzazione posizionale**:

1. Si costruisce un vettore binario casuale $m \in U(\mathbb{B}^m)$, chiamato *maschera*.
2. Si procede quindi a scambiare gli elementi seguendo la

regola seguente:

$$\begin{cases} \text{I simboli in posizione } i \text{ restano invariati} & m_i = 1 \\ \text{I simboli in posizione } i \text{ vengono scambiati} & m_i = 0 \end{cases}$$

13.2.9 Mutazione

Definizione 13.2.8 (Mutazione). L'operatore di **mutazione** modifica un individuo per generarne uno simile. Tipicamente procede scorrendo la codifica ξ simbolo per simbolo e decide quindi con probabilità π_m se modificarlo o meno.

13.2.10 Ammissibilità

Mutazione e crossover possono creare alcune codifiche, quando la relazione tra codifiche e soluzioni non è completa, che non sono realmente ammissibili. Esistono 3 approcci principali per contrastare questo problema:

Rappresentazione e operatori speciali

Si cerca di creare solo rappresentazioni che producono (o quasi) codifiche ammissibili o alternativamente creare operatori che conservano l'ammissibilità, ma questi metodo abbandonano l'idea dell'astrazione dal problema specifico e tendono ad avvicinarsi parecchio a dei metodi di scambio e ricombinazione.

Procedure di riparazione

Come è intuibile, modifica la soluzione sino a che essa diviene ammissibile, ma questo porta a una forte polarizzazione verso le codifiche ammissibili ed introduce uno sbilanciamento verso le soluzioni ammissibili più facili da ottenere utilizzando la procedura di "sanitizzazione" che si ha realizzato.

Funzioni di penalità

Definizione 13.2.9 (Codifiche ammissibili). Chiamiamo codifiche ammissibili le codifiche che corrispondono a soluzioni ammissibili.

Questo metodo si basa su una **metrica di inammissibilità** ϕ del tipo:

$$\phi(x) = \begin{cases} 0 & x \in X \\ > 0 & x \notin X \end{cases}$$

Se i vincoli del problema sono espressi da uguaglianze o disuguglianze, si può definire $\psi(x)$ come la somma pesata delle loro violazioni.

La fitness ϕ tipicamente è un'opportuna combinazione della funzione obbiettivo e della misura di inammissibilità.

Estensioni comuni sono:

Definizione 13.2.10 (Penalità assoluta). Date due codifiche ξ e ξ':

- Se entrambe sono ammissibili, è meglio quella con f minore.
- Se una sola è ammissibile, quella ammissibile è meglio dell'altra.
- Se entrambe sono inammissibili, è meglio quella con ψ minore.

Definizione 13.2.11 (Penalità legata al costo di riparazione). Data una procedura che, applicata a x, produce una soluzione $x' = r(x)$ ammissibile:

$$\phi(\xi) = \frac{1}{f(x')} = \frac{1}{f(r(x(\xi)))}$$

cioè si usa il valore della funzione obbiettivo nella soluzione "riparata".

Definizione 13.2.12 (Penalità uniforme). Le codifiche inammissibili hanno una penalità fissa:

$$\phi(\xi) = \begin{cases} \frac{1}{f(x(\xi))} & \text{Per le codifiche ammissibili} \\ \frac{1}{f(x(\xi))+\alpha} & \text{Per le codifiche inammissibili} \end{cases}$$

Definizione 13.2.13 (Penalità proporzionale). La penalità cresce con la violazione:

$$\phi(\xi) = \frac{1}{f(x(\xi)) + \alpha\psi(x(\xi))}$$

Tarare la penalità

Sperimentalmente si deduce che conviene usare la penalità minima efficace: se essa è troppo bassa non si trovano soluzioni ammissibili, e se è troppo alta si rimane confinati in una regione ammissibile.

Sono stati proposti un numero di metodi per tarare il parametro α:

Metodi dinamici: α cresce nel tempo.

Metodi adattativi: α cresce se nella popolazione prevalgono le codifiche inammissibili, cala se prevalgono quelle ammissibili.

Metodi evoluzionistici: ogni individui codifica anche α, che viene selezionata e raffinata dall'euristica insieme alla soluzione.

13.3 Algoritmi memetici

Definizione 13.3.1 (Meme). Unità base di informazione culturale riproducibile

Gli algoritmi memetici si ispirano al concetto di meme e combinano:

1. Gli operatori genotipici che manipolano le codifiche.
2. Gli operatori fenotipici che manipolano le soluzioni.

Le soluzioni vengono migliorate con scambi prima di essere ricodificate.

13.4 Strategie evoluzionistiche

Sono molto simili agli algoritmi genetici, le principali differenze sono che:

1. Le soluzioni sono codificate in vettori di numeri reali.
2. La popolazione è ridotta a un singolo individuo che ne produce λ (si preferisce spesso usarne un piccolo numero μ).
3. Esistono due varianti fondamentali:
   (a) Nella strategia $(1+\lambda)$ i nuovi individui competono col progenitore e il migliore di tutti sopravvive.
   (b) Nella strategia $(1,\lambda)$ i nuovi individui competono e il migliore fra loro sostituisce il progenitore, anche se quello lo domina.
4. L'operatore di mutazione somma un disturbo casuale con distribuzione normale a media nulla.
5. Non esisteva in origine l'operatore di crossover, ma attualmente viene utilizzato anche in questi algoritmi.