29/04/2021

**RICERCA LOCALE**

**CLASSIFICAZIONE TECNICHE RISOLUTIVE**

**CATEGORIE**:

* ESATTE: trovano la soluzione ottima
* EURISTICHE: trovano una soluzione buona
* COSTRUTTIVE: partono dalla soluzione vuota e ne costruiscono una passo-passo --> es. parto da un grafo con nessun nodo colorato, poi coloro un primo nodo, un secondo e così via
* SELETTIVE: esplorano lo spazio delle soluzioni complete e si muovono in questo spazio

**TECNICHE**:

* BACKTRACKING (costruttiva + esatta)
* ENUMERAZIONE (selettiva +esatta)
* GREEDY (Costruttive - euristiche)
* Immagine che contiene gara di atletica

  Descrizione generata automaticamenteRICERCA LOCALE (selettive – euristiche)

**DIFFERENZA TRA COSTRUTTIVA E SELETTIVA**:

Costruttive --> lavorano sul search tree,

Albero di ricerca

(search tree)

Selettiva --> lavorano sullo search space



(= insieme delle foglie del search tree).

Spazio di ricerca

(search space)

SELETTIVE

ESATTE

COSTRUTTIVE

**BACKTRACKING**

**ENUMERAZIONE**

**RICERCA LOCALE**

**GREEDY**



EURISTICHE

**RICERCA LOCALE**

--> Lavora sullo spazio di ricerca

**FUNZIONAMENTO**:

* Lo spazio lo dobbiamo definire noi; non è sempre dato esplicitamente dal problema (si può decidere di usare uno spazio diverso da quello delle soluzioni del problema vero e proprio

Ottenendo quindi una rappresentazione anche implicita diversa)

* Si parte da una **soluzione iniziale** (*initial solution*): può essere o trovata con un metodo - tecnica (tipicamente Greedy) oppure può essere anche casuale (ci interessa una tecnica veloce – random ~ *initial solution procedure*, il lavoro lo farà poi la ricerca locale; es. Graph Coloring per ogni nodo tiro un colore a caso) --> così ovviamente la soluzione iniziale può avere violazioni

**CARATTERISTICHE**:

* **Spazio di ricerca** – *search space*
* **Intorno della soluzione** – *neighborhood* = intorno/insieme di soluzioni che sono vicine alla mia ottenibili con una con una singola modifica (es. Graph Coloring = cambiare il colore di un nodo quindi io definisco i vicini come tutti quelli che sono identici alla soluzione iniziale ma differiscono del colore di un nodo).

Questo è ovviamente molto dipendente dal problema e quindi ogni problema avrà la sua definizione di neighborhood (anche diversi per uno stesso problema).

* La ricerca locale esplora una parte dell'intorno (non necessariamente tutto) (a seconda della specifica tecnica – alcune lo esplorano tutto, altre ne guardano solo una frazione)
* Quindi seleziono in qualche modo un vicino: valuto sia l'**esplorazione** (*exploration*) cioè quanta parte esploro e quale parte, sia come scelgo il vicino sono tutte lasciate alla specifica tecnica (**criterio di accettazione**).
* L'intuizione è che io scelgo il migliore --> se qui ho un certo numero di violazioni scelgo quello che mi riduce il numero di violazioni es. se ho due nodi dello stesso colore prendo uno dei due e gli cambio colore
* Ho quindi un criterio di accettazione (*acceptance*) = criterio che mi dice se il vicino scelto effettivamente è una buona scelta; anche qui questo può essere “se migliorativo”.
* Quindi ci si sposta al nuovo punto che ha passato l’accettazione (prima freccia rossa)
* Il concetto di intorno è simmetrico quindi la soluzione precedente è nell'intorno di quella appena trovata
* Riseleziono un vicino e se decido di accettarlo mi sposto a questo e così via
* Se non lo accetto rimango lì e continuo a esplorare da li
* In tutte queste scelte sono guidato da una **funzione di costo** (*cost function*); quindi a ogni soluzione assegno un costo (es. funzione obiettivo del mio problema); posso farmi guidare dai più criteri artificiali che mi permettono di fare delle scelte migliori
* C’è un **criterio di arresto** (*stop criteria*) che dicono che a un certo ci si deve fermare alla **soluzione finale** (final solution)

Alcune di queste caratteristiche (in grassetto) sono molto problem dependent - specifiche, altre più generiche (infatti le tecniche di ricerca locale prendono anche il nome di metaeuristiche).

ESEMPI CRITERI DI ARRESTO:

* quando ho passato un certo numero interazioni (più banale - tipico) (es. dopo 100.000 interazioni)
* quando ho passato un certo numero interazioni senza miglioramenti (se migliora la funzione di costo gli do ancora fiducia e vado avanti --> Stagnation Detection ~ rilevamento della stagnazione



INITIAL SOLUTION



FINAL SOLUTION



**ESEMPIO N\_QUEENS**

*SEARCH SPACE*

* Immagine che contiene shoji, cruciverba, piastrellato

  Descrizione generata automaticamenteS1: uno spazio potrebbe essere quello delle scacchiere (non nostra), quindi una **matrice booleana** (si-no, la regina c’è o non c’è)
* S2: **spazio delle assegnazioni** (quello usato per il backtracking) ovvero prendo un vettore che è lungo quante sono le regine e scrivo la posizione delle regine (spazio più piccolo perché non ci stanno soluzioni con regine nella stessa colonna o con più regine (es. 5 o 6), quindi sono incluse solo le soluzioni ammissibili)

Immagine che contiene testo, shoji, cruciverba

Descrizione generata automaticamenteImmagine che contiene shoji, interni, piastrellato, tegola

Descrizione generata automaticamente

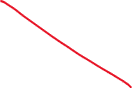
* S3: **spazio delle permutazioni**, ovvero un vettore all’interno del quale ci sono solo permutazioni (quello usato per l’enumerazione) (questa soluzione, ad esempio, non è ammissibile) --> spazio più piccolo

Immagine che contiene shoji, cruciverba

Descrizione generata automaticamenteImmagine che contiene shoji, interni, piastrellato, tegola

Descrizione generata automaticamenteImmagine che contiene shoji

Descrizione generata automaticamente



*NEIGHBORHOODS*

* Certe mosse vanno bene per certi spazi e non per altri
* Caratteristica fondamentale del neighborhood è che lo spazio sia connesso ovvero che io facendo una mossa posso andare da qualunque punto a qualunque punto

Immagine che contiene tennis, racchetta, gara di atletica, corte

Descrizione generata automaticamente

* Immagine che contiene testo, shoji, orologio

  Descrizione generata automaticamente**Move**: prendo una regina e la sposto (in un'altra riga) --> *per assignments*

Data una soluzione i suoi vicini sono tutti quelli che hanno un solo valore diverso --> n-1 vicini

Immagine che contiene testo, lavagnabianca

Descrizione generata automaticamenteQuindi nel complesso ho **n\*(n-1) vicini**

Questa mossa va bene per S1, S2, non S3 che esce dallo spazio

* Immagine che contiene testo, lavagnabianca, ricevuta

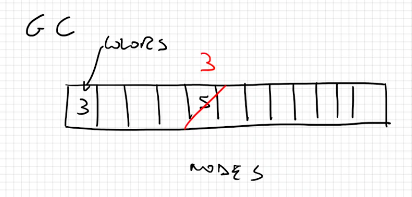
  Descrizione generata automaticamente**Swap**: scambio tra di loro due regine --> *per permutations*

Immagine che contiene tennis, racchetta, corte, giocatore

Descrizione generata automaticamenteQuesta mossa va bene per S3, non va bene per S2 (non per valori uguali in quanto sarebbe inutile, non avrebbe senso scambiare valori uguali) perché lo spazio di ricerca non è connesso sotto questa mossa, ovvero quando faccio swap fra assegnazioni non esco mai da un pezzo dello spazio (es. se non c’è un 1 nel vettore allora non ce lo metto mai, non riesco a scrivere valori che non ci sono)

* Altre mosse: mosse che scambiano 3 regine, mosse che ribaltano un pezzo della soluzione, etc. --> li è la fantasia del progettista

Immagine che contiene sport, gara di atletica

Descrizione generata automaticamente**ESEMPIO GRAPH COLORING**

*Spazio di ricerca*: ho un insieme di nodi e a ciascuno di questi do un suo colore

*Mossa*: Move che nel GC si chiama **recolor**, quindi prendo un nodo e do un colore e lo ricoloro

Anche qui esistono mosse complesse, esempio cambio il colore di un certo numero di nodi

Anche questo spazio di ricerca è connesso

Il numero di vicini per questo spazio è **n\*(c – 1)** (visto che ogni nodo ha c-1 colori di ricolorazione)

Ovviamente noi non consideriamo la soluzione stessa come suo vicino

Tutti questi vicinati che abbiamo visto sono simmetrici, infatti se io ho una mossa che da un punto mi porta ad un altro esiste anche la mossa inversa (tipico)

**COME SCEGLIERE LO SPAZIO DI RICERCA**

* + Deve contenere almeno una SO se parliamo di problema di ottimizzazione, se di ricerca almeno una SA
  + Deve essere il più possibile piccolo (perché ci vuole meno tempo per visitarlo) (ma non è così pressante come l’enumerazione in cui devo visitare tutto)
  + Deve essere facile trovare la soluzione iniziale, eventualmente random o Greedy

Immagine che contiene gara di atletica, sport

Descrizione generata automaticamente**COME ESPLORIAMO L’INTORNO**

--> varie tecniche per decidere quale porzione dell’intorno considerare:

(zoommo l’intorno)

* Cercare il miglior vicino possibile: faccio un’enumerazione del vicinato (delle mosse, visto che a ogni mossa corrisponde un vicino) ~ faccio una serie di first e next (FM = first move, NM = next move); Una volta finito memorizzo la BM = best move

Immagine che contiene testo, gara di atletica, sport

Descrizione generata automaticamente

- **n\_regine** (es. n = 4) con vicinato swap e spazio permutazioni, le mosse le enumero in base alle regine coinvolte; es. dico che la prima mossa è **<0 ,1>** che fa lo swap delle regine in posizioni 0 e 1: è un’enumerazione su uno spazio di dimensione 2 invece che N !

*q1, q2*

<0, 1> (<1, 0> non ci serve perché sarebbe identico)

<0, 2>

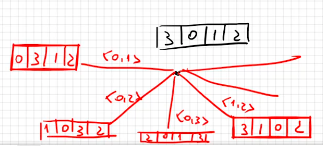
<0, 3>

<1, 2>

<1, 3>

<2, 3>

Es. Ha 6 vicini (in rosso, con mosse e vicini che ottengo)

Immagine che contiene gruccia

Descrizione generata automaticamente

Ho coppie

- **GC**, la mossa è prendere un nodo e cambiargli colore

Immagine che contiene shoji

Descrizione generata automaticamente

Es. di soluzione

E tutte le possibili mosse hanno i due possibili attributi il nodo e il nuovo colore

--> qui le mosse disponibili sono dipendenti dallo stato (es, da qui ad esempio prendere il nodo 0 che ha il colore zero e dargli di nuovo il colore zero non ha senso)

*N, C*

<0, 1>

<0, 2>

<1, 0>

<1, 2>

<2, 0>

<2, 1>

<3, 0>

…

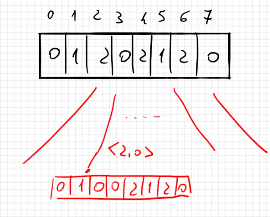
Immagine che contiene shoji

Descrizione generata automaticamente

È come se contassi su due cifre: prima faccio muovere la cifra di dx (meno significativa) e poi quella di sx (più significativa), ognuno con le sue regole

**ESEMPIO REGOLE:**

* Es n\_regine: la cifra meno significativa non deve mai andare sotto quella più significativa --> quindi da 0, 3 eravamo passati a 1, 2
* Es GC: non devo mettere lo stesso valore dello stato

**ESEMPIO VICINI:**

1. **STEEPEST DESCENT**

--> strategia di scegliere sempre il migliore ma accettare solo mosse migliorative

- enumero tutti i vicini e cerco il migliore **(*best neighbor*)**

-stopquando **not improving in neighborhood**

Quindi esploro tutto l’intorno, se trovo una mossa migliorativa la faccio e mi sposto altrimenti mi fermo, quindi in. Questa tecnica il criterio di arresto è un criterio basato sullo stato – situazione corrente**.**

Quello in cui mi fermo non è detto in generale che sia l’ottimo globale ma è un **ottimo locale** (tipicamente ce ne sono tantissimi in un problema)

Immagine che contiene testo, gara di atletica, sport

Descrizione generata automaticamenteA volte si può incorrere in un tie-break, ovvero io posso avere più migliori (es. stesso valore di cost function ~ es. numero di violazioni, qui nel GC numero di coppie adiacenti dello stesso colore, n\_regine = numero di regine che si attaccano)

(in blu numero di violazioni). Qui quale scelgo tra i tre con 7 violazioni? Scelgo con probabilità uniforme

--> randomizzazione pesata

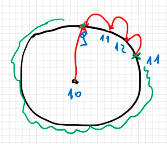
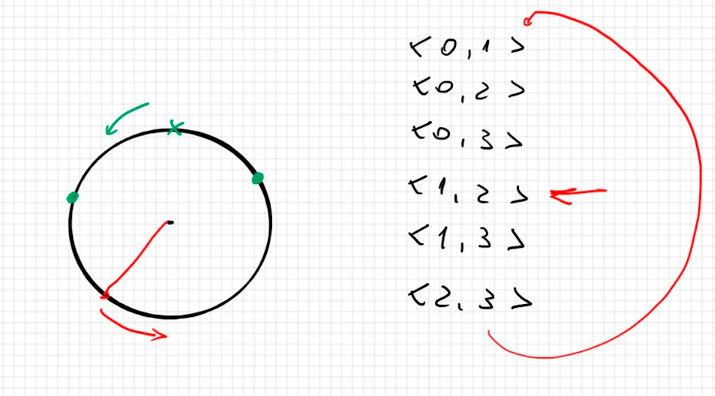
1. **FIRST IMPROVING DESCENT**

--> tecnica alternativa migliorativa del steepest descent (che è molto lento perché esploro l’intero intorno 🡪 costoso)

Invece di esplorare tutto, ho sempre first e next, quando trovo una che ha costo minore (qui il 9) mi ci sposto e inizio a esplorare l’jntorno di questo nuovo stato, con l’idea che non vale la pena esplorare tutta la parte verde visto che ho avuto un miglioramento:

- esplora l’intorno finche’ non trovi una mossa migliorativa (***first improving neighbor***), appena la trovi la fai e riparti da li (e poi **stop at local minimum** (come sopra, stessa cosa))

- PROBLEMA DI BASE di FARINESS: se faccio un’enumerazione delle soluzioni partendo sempre dalla stessa, i first improving magari vengono scelti sempre nello stesso verso (magari scelgo lavoratori con indice nell’istanza basso) --> per risolverlo tiro una mossa a caso e a partire da quella soluzione risolvo il problema da li e poi circolarmente dall’ultimo ritorno alla prima

****

Ho più speranze a questo punto di trovare un 8 nell’intorno del 9 invece che dell’11

1. **RANDOM DESCENT**

--> mentre con il first improving descent esploro tutte mosse simili tra loro quindi tutti stati vicinissimi tra loro, potrei cambiare completamente strategia e quindi fare delle mosse ***random***, e se questa è migliorativa la faccio, altrimenti se non lo è no --> tiro valori qualunque degli attributi e se trovo migliore cambio

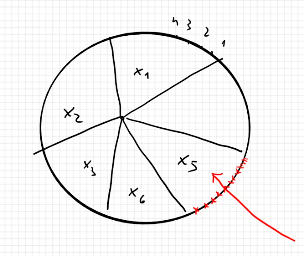
Immagine che contiene testo, gara di atletica, sport

Descrizione generata automaticamente

Qui 2 CRITERI DI STOP (diverso dagli altri perché non posso fermarmi al minimo locale in quanto non ho modo di verificare, perché se tiro a caso non so mai se gli ho tirati tutti):

* Numero di iterazioni (= mossa random anche se non l’accetto)
* Numero di iterazioni senza muovermi --> stagnation detection = idle iterations

1. **MIN CONFLICT**

**-->** esplora una parte dell’intorno: es. se immaginiamo che lo spazio è quello di avere tante variabili (x1, x2, x3, …) e i loro valori (1, 2, 3, 4), (come nodi e colori del GC), quando ho uno spazio di questo tipo una cosa tipica è quella di fare una via di mezzo ovvero di fare una random variable, ovvero tiro a caso una variabile (es. x5) e poi esploro tutti i possibili valori = via di mezzo tra steepest descent e random descent

* Se io da una soluzione faccio qualunque di queste tecniche (es. best, first improving, etc.), e trovo una soluzione che **costa anche lei 10**, questa è una mossa laterale = **side ways**. L’esperienza sperimentale insegna che queste è meglio farle perché mi sposto; è possibile che io abbia situazioni con tante mosse senza miglioramento e poi finalmente trovo una mossa migliorativa (es. 9). Ci sono zone dello spazio di ricerca che sono cosiddetti plateau, quindi mi conviene spostarmi all’interno di questo plateau fino a trovare un punto da cui si esce. (così ho più speranze di uscire facendo sideways) --> questo si può applicare a tutti i nostri metodi (radom non-ascent)

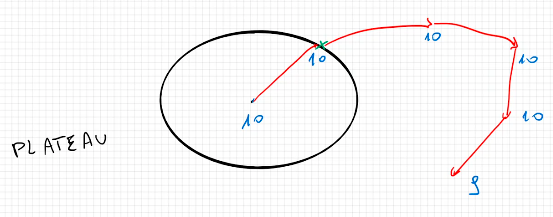


Immagine che contiene testo, lavagnabianca

Descrizione generata automaticamenteEsiste infatti tutta una serie di tecniche sorelle di quelle viste che al posto di descent faccio il **non-ascent**

(random/first\_imrpoving/best non ascent)

ESEMPIO BEST NON-ASCENT

--> cerco la soluzione migliore e se non trovo alcuna soluzione migliorativa accetto anche una soluzione a costo pari, quindi quello che trovo non è un minimo locale stretto, e prendo e vado li

A questo punto non mi fermo più con il steepest descent, perché se io ho una soluzione di costo pari da li ne avrò un'altra di costo pari che è quella da cui sono partito.

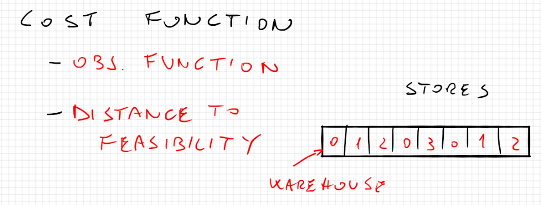
In questo caso quello che faccio è fissare un numero di iterations o idle iterations, o anche tutte e due assieme

**COST FUNCTION**

--> solitamente è la funzione obiettivo del problema

--> in questi problemi possiamo considerare anche soluzioni NA, ovvero che hanno delle violazioni

Per il Waehouse location problem la cost function è:



* Object function = somma costi
* **Distance to feasibility** = per il calcolo è una scelta del progettista, e può essere il numero di warehouse che sta dando troppo ma anche la quantità di troppo che sta dando (HA PESO MAGGIORE (es. x10000) rispetto a object function,che vedo solo a distanza 0)

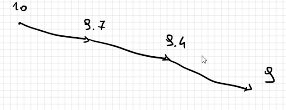
Se un warehouse per es sta dando due volte quello che da, potrebbe non esistere alcuna mossa che mi fa perdere quella violazione, perché sono molto lontano.

Voglio evitare se possibile una soluzione

Immagine che contiene testo, lavagnabianca

Descrizione generata automaticamente

Se si deve fare si fa ma sarebbe meglio che si avesse



Quindi non conto quanto warehouse sono fuori ma conto la somma di quanto sono fuori (primo caso fuori di 3, secondo di 1.9)

5/05/2021

**RIASSUNTO**

LOCAL SEARCH:

* Incomplete and selective technique
* Good in practice
* Experience in Uniud

INGREDIENTI:

* Definire il Search space
* Neighborhood
* Cost function (objective function, distance to feasibility, auxiliary functions ~ alcune volte non riusciamo a valutare che una soluzione è migliore di un’altra perché la funzione obiettivo non ci aiuta; es. sforo di tanto o sforo di poco, sforo ovunque ma in maniera diversa)
* Initial solution strategy (random ~ most common, Greedy, meno preferibili delle precedenti per un discorso di minimi locali)
* Stop criteria (local minimum, number of iterations, stagnation detection 🡪cioè mi accorgo che è da un pò di iterazioni che non miglioro)
* Acceptance (descent = accettare solo improving, non-ascent = accetto anche sideways)
* Exploration (best – molto lenta, first improving – non aspettare e parti da li, random – cerco di non guardare mosse simili tra loro, faccio campionamenti e perdo meno tempo, mixed – in cui si esplora una parte del vicinato partendo random da una variabile, etc.)

TECHNIQUES:

* HILL CLIMBING = LOCAL SEARCH (HC)
* Tecniche di ricerca locale con idea che per uscire da un minimo locale si accettano anche **mosse peggiorative** per sperare in un miglioramento:
  + SIMULATED ANNEALING (SA)
  + TABU SEARCH (TS)
  + Hybrids
  + Random walk
  + …

**SIMULATED ANNEALING**

- IDEA: Quando la temperatura è alta (all’inizio) si fanno anche molte mosse peggiorative, quando questa cala(nel tempo) se ne fanno meno, quindi alla fine facciamo solo mosse migliorative laterali. Quindi:

- parto da una RANDOM INITIAL SOLUTION

Immagine che contiene testo

Descrizione generata automaticamente- RANDOM MOVE (a ogni iterazione faccio una mossa casuale 🡪 cioè scelgo un vicino a caso)

Metodo per scegliere 2 numeri random ma che non siano uguali

- ACCEPTANCE CRITERION (il discorso di accettazione si basa sul = costo soluzione finale (S1) – costo soluzione iniziale (S0) = delta della funzione di costo)

Immagine che contiene testo

Descrizione generata automaticamente

Se <= 0 allora viene sempre accettata

Se > 0 la mossa viene accettata con probabilità = con T = temperatura

(ESEMPIO = 3, T = 1.8, = 0.18 quindi accetterò la mossa con questa probabilità, quindi devo tirare un numero a caso nell’intervallo [0, 1] e se il numero è < 0.18 accetto la mossa, altrimenti non la accetto)

Immagine che contiene filo, disegnoatratteggio, giorno

Descrizione generata automaticamente

Quindi accetto una mossa:

* Con probabilità più alta tanto più alta è la temperatura
* Con probabilità più bassa più bassa è la temperatura

Quindi:

* All’inizio ne accetto tante, poi via via sempre meno
* Più la mossa è cattiva (ΔF grande) e più bassa è la probabilità di accettarla

COOLING: modalità di evoluzione della temperatura che nel tempo cala secondo la relazione

--> T = a \* T con a = cooling rate (es. 0.9, fattore che smorza, quindi <1)

PSEUDOCODICE

Immagine che contiene testo

Descrizione generata automaticamente

Calcolo il delta F = costo nello stato facendo la mossa (applicando m) – costo stato



Tf = temperatura minima --> criterio di arresto = arrivarci

Es. Tmax = 100, Tmin = 1

Temperatura iniziale To (devo fissarlo)

s = stato random corrente nello spazio di ricerca S

sbest = stato migliore memorizzato (= stato corrente inizialmente)

ns = contatore mosse che faccio a ciascuna temperatura

Ns = parametro (tipicamente faccio Ns = 1000 iterazioni)

m = mossa random nello stato s dal vicinato N

**VARIANTI**

1) Per velocizzare la parte iniziale in cui vengono fatte molte mosse (per smuovere la soluzione), io scendo di temperatura non solo se ho fatto un certo numero di iterazioni, ma anche se ho accettato un certo numero di mosse

(na e Na numero di mosse accettate e parametro massimo di mosse accettabili)

Immagine che contiene testo

Descrizione generata automaticamente



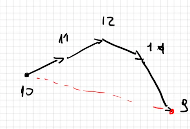
Vedremo framework



2) Altre varianti per formula esponenziale e random che costano e smorzamento/abbassamento temperatura

**TABU SEARCH**

- IDEA: a ogni iterazione scelgo la mossa migliore, non vado a caso, ma faccio la mossa migliore possibile anche se peggiorativa (es. qui sotto steepest descent si fermerebbe non vedendo miglioramenti che però più avanti ci sono)



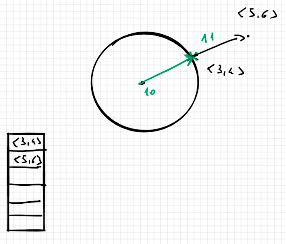
- PROBLEMA: in molti casi c’è il rischio che la mossa migliore sia quella di tornare indietro se sto peggiorando --> per risolverlo indico delle mosse tabu ovvero dico che certe mosse non si possono fare

MOSSE TABU

- Ho una tabu list, in cui scrivo tutte le mosse che non si possono fare (son stati che mi fanno pensare a direzioni sbagliate)

ESEMPIO N REGINE

Ho mosso 3, 4 allora scrivo <3, 4> e non posso tornare indietro, e così via



NOTE:

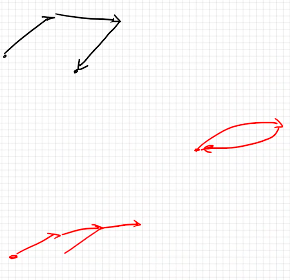
1. Se ho fatto prima una mossa e poi un’altra, se questa mossa è rimasta nella tabu list io sto proibendo anche una mossa che non è un tornare indietro ma va in uno stato nuovo

Es.

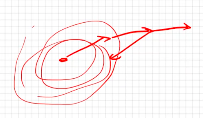




Io, comunque, quello lo voglio evitare perché se voglio muovermi da un minimo locale non solo voglio non tornare indietro ma anche allontanarmi da questo (per via del suo bacino di attrazione)



Quindi voglio evitare di muovermi caoticamente attorno al minimo locale e proibendo varie mosse (**proibisco le inverse delle mosse della tabu list, non le mosse stesse**)riesco a scappare --> a ogni iterazione scelgo la migliore mossa che non sia tabu



LUNGHEZZA TABU LIST = FISSA (e deve essere quella giusta, il giusto compromesso, devo trovarla), con a ogni iterazione una mossa che entra e una che esce

**CARATTERISTICHE**

- TABU LIST

- ASPIRATION CRITERIA

--> quando esploro l’intorno anche se una mossa è tabu, io comunque le valuto quelle tabu e se quelle tabu sono talmente buone da soddisfare un certo criterio di aspirazione, cioè danno un miglioramento di almeno ‘x’, allora le faccio lo stesso

**VARIANTE (implementata nel nostro framework)**

- DYNAMIC TABU LIST: fisso una lunghezza minima e massima della tabu list

--> ogni volta che una mossa viene eseguita va nella tabu list e gli viene inserito insieme alla mossa anche la durata in cui deve stare nella tabu list tra quelle che sono lunghezza minima e massima

Immagine che contiene testo, shoji, orologio

Descrizione generata automaticamente

In realtà nella colonna di dx indico l’iterazione a cui deve uscire (ci sono mosse che entrano-escono prima di altre)

**ATTENZIONE NEL GENERARE LE MOSSE RANDOM**

Suppongo di avere la seguente strategia random per q1, q2 (per generare una mossa)

Immagine che contiene testo

Descrizione generata automaticamente

N = numero di regine (quindi q1 avrà n-2 essendo n-1 l’ultima)

Devo fare come random q2 una che sia più grande di q1 (quindi q1+1 e n-1)

Questa strategia non è una buona mossa perché supponendo che Random sia uniforme, che mi generi tutto con le stesse probabilità, io ho le seguenti probabilità (in verde)

Immagine che contiene testo, lavagnabianca

Descrizione generata automaticamente



Quindi io ho maggiore probabilità di scegliere numeri alti

Per risolvere questo disequilibrio è quello di tirare due numeri qualunque a patto che siano diversi ed eventualmente scambiarli, come segue:

Immagine che contiene testo

Descrizione generata automaticamente

Qui problema è che q1 e q2 devono essere diversi

Immagine che contiene testo

Descrizione generata automaticamente

Questo però è un po’ inefficiente, vorrei evitare cicli, quindi sposto tutti in avanti non considerando q1:

Immagine che contiene testo, lavagnabianca

Descrizione generata automaticamente

6/05/2021

**EASY LOCAL ++**

Il **FRAMEWORK** che utilizziamo per fare ricerca locale prende il nome di **Easylocal ++**, noi usiamo la versione 3, con installazione di boost, è un SW header-only

COMANDO TERMINALE:

--> sudo apt install libboost-all-dev (ma tanta roba)

--> sudo apt install libboost-program-options-dev

--> sudo apt install git

--> git clone <https://bitbucket.org/satt/easylocal-3.git>

Quindi dovrò includere Easylocal++

**ARCHITETTURA**

Come si usa? (avrò un workflow di easy local, anzi mi conviene partire da esempi fatti) (options = coldspot, altrimenti hotspot --> nel workflow)

Ho tante classi combinate tra loro.

Ci sono 4 livelli di oggetti – **basic classes (templates)**:

1. Classe di INPUT
2. Classe di OUTPUT --> come vuole vedere l’utente la soluzione
3. Classe STATE --> un punto nello spazio di ricerca, soluzione (ho qualcosa che me lo traduce in output e viceversa) (viene sporcato per eventuali aggiunte-ridondanze utili lo state, non l’output)
4. MOVE --> rappresenta una mossa (tipicamente con pochi attributi, 2, 3 o 4, come q1 e q2 degli esempi precedenti)

**Classi astratte – helpers 🡪 dobbiamo scrivere la sottoclasse:**

1. OUTPUT MANAGER (che mi traduce da output a stato e viceversa)
2. STATE MANAGER (trova soluzione iniziale, calcola costo dello stato) --> posso averne uno
3. NEIGHBORHOOD EXPLORER (gestisce uno specifico intorno) --> e averne tanti di questi visto che posso avere più mosse
4. COST COMPONENT (vettore di componenti e per ognuna state manager chiede il costo)
5. DELTA COST COMPONENT (calcola il costo data una certa mossa) --> questa dipende dal neighborhood

**Classi di alto livello – runner:**

1. SA 🡪 simulated anealing
2. TS 🡪 Tabu Search
3. HC
4. …

**Solver** (orchestra i vari runner) **/tester**

**ESEMPIO N\_QUEENS** (vedi codice)

**BASIC CLASSES:**

- INPUT (già vista) – chess board con dentro intero

- OUTPUT – scacchiera = matrice nxn di caratteri

--> queste due stanno nella classe QUEENS DATA

- STATE – permutazioni (oppure anche assegnazioni)

- MOVE – swap

--> queste due stanno nella classe QUEENS BASICS

**PER PROGETTO POSSO PRENDERE N\_REGINE E MODIFICARLO MA C’E’ A DISPOSIZIONE ANCHE IL EL3SEED PROJECT**, ovvero un file vuoto con tante cose da fare (hanno un throw che dice che una data cosa non è stata ancora implementata, ovvero le cose elencate in precedenza)

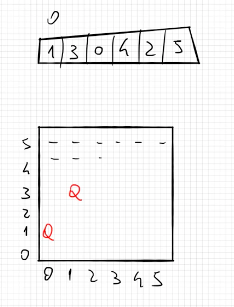
**HELPERS:**

- OUTPUT MANAGER (ha i due metodi OUTPUTSTATE E INPUTSTATE)

- STATE MANAGER (RANDOMSTATE, CHECKCONSISTENCY)

- NEIGHBORHOOD EXPLORER (FIRSTMOVE e NEXTMOVE che servono per esplorare l’intero vicinato, coldspot BESTMOVE non lo vedremo, sarà dentro il framework che chiama first e next, RANDOMMOVE che genera la mossa random, MAKEMOVE per farla = modifica lo stato eseguendo la mossa, FEASIBLEMOVE che mi dice che la mossa è fattibile)

ES. Outputstate



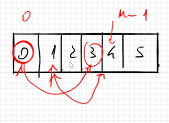
**COME FARE UNA PERMUTAZIONE RANDOM**

--> prima si scrive una permutazione base, e poi si fa uno shuffle del vettore per creare vettore casuale e uniforme

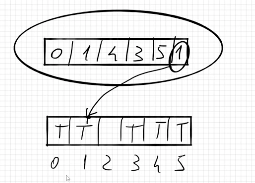
Immagine che contiene shoji, orologio

Descrizione generata automaticamente

Quindi tiro un numero a caso tra 0 e n-1, quindi scambio lo zero con questo numero; vado al successivo e lo scambio con numero generato a caso, e così via

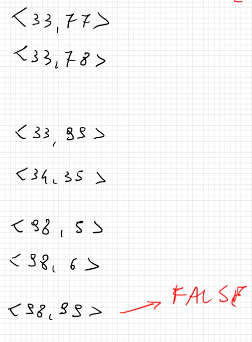


**CHECK CONSISTENCY**



Qui mi dice che 1 è già stato taggato --> non è consistente

ESEMPI NEXTMOVE



ESEMPIO ISTANZA 100 N REGINE

Per usare il programma su un’istanza scrivo la seguente riga di codice:

./nqueens –main::instance queens8.txt --> questo ‘main::instance’ è l’istanza, e il filename seguente va dentro il parametro di tipo Parameter <string> convertibile a string, che si chiama instance

Immagine che contiene testo

Descrizione generata automaticamente

Verifico distribuzione delle mosse random

Ho 4950 vicini, quante volte voglio tirare? --> 100 mosse (tiro 4950 x 100 mosse e vedo al distribuzione rispetto a queste che sono generate da first e next)

Immagine che contiene testo

Descrizione generata automaticamente

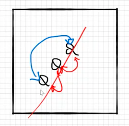


--> vedo valori abbastanza casuali e ci stanno (es. la mossa <0, 41> è stata tirata 1.3 volte rispetto alla sua normale probabilità, ma ci sta, può succedere, se utilizzassi la versione sbagliata di random state come visto in precedenza la situazione è nettamente squilibrata in favore delle ultime mosse)

Check move independence --> controllo che non ci siano due mosse che fanno la stessa mossa

(le verifica tutte e mi dice che non ci sono mosse ripetute (che fanno lo stesso stato) e mosse nulle, ovvero mosse che non smuovono niente)

NOTA: In figura ci sono 3 violazioni



**DELTA COST COMPONENT N QUEENS**

Immagine che contiene testo, lavagnabianca

Descrizione generata automaticamente



Voglio spostare le regine come in rosso; non vado a vedere se le due regine in blu si attaccano tra di loro, non ci interessa dal punto di vista del delta perché se si attaccavano prima lo faranno anche dopo e viceversa

Quindi vedo tutti quelli che attaccavano le due sottolineate e li tolgo (costo--), e aggiungo invece i costi di quelli rossi (costo++)



--> Ho un ciclo con quattro if dentro

COSTO = -2000



--> questo perché: ho obj = 0 perché non ce l’ho, e le violazioni lui le pesa 1000 --> -2 (di default) \* 1000 = 2000

--> quindi questa è una mossa che migliora ovvero fa togliere due violazioni

6 --> statistica delle mosse e da tutte quelle possibili mi mette le migliorative, peggiorative e sideways

Immagine che contiene testo

Descrizione generata automaticamente

12/05/2021

SCELTE 3, 4 E 9 MENU:

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

9 --> lui internamente valuta il costo delle mosse e quindi valuta il delta che ho scritto rispetto alla funzione di costo

**SEED**







Lo rifaccio e ottengo lo stesso risultato perché sto dando sempre lo stesso seed.

Quindi per fare debugging meglio avere lo stesso seed in modo tale da essere in grado di riprodurlo.

Posso passare la soluzione iniziale direttamente grazie al parametro definito nel codice e scritto a riga di comando

**Parameters : vedi lezione 12 maggio al minuto 14 circa della prima parte**

**BUS DRIVER SCHEDULING**

- Shift

- Task

- Quindi ci sono degli shift, ciascuno dei quali ha un costo e un insieme di task che copre; i task devono essere tutti coperti e devo trovare la soluzione che copre certi shift per fare questo al costo minore possibile

**SEARCH SPACE**

Soluzione più semplice --> VETTORE DI BOOLEANI (ogni shift ha un elemento T/F)

Lo Stato avrà quindi un vettore di booleani più eventualmente altre cose come acceleratori per dati ridondanti (es. chi copre chi, etc.)

Spazio di ricerca = 2^(n° di shift)

**NEIBHBORHOODS**

Per definire la relazione di intorno = suoi vicini, faccio dei **FLIP** = cambio di valore uno ad uno gli elementi del vettore

(qui ho l’unico attributo dello shift che cambio)

Immagine che contiene testo, sport

Descrizione generata automaticamente

In tanti problemi è il vicinato più ragionevole, nel nostro caso ha il problema che noi vogliamo il vicinato tale per cui esistano dei percorsi di discesa da una soluzione cattiva ad una buona di questo tipo:

Immagine che contiene testo, filo

Descrizione generata automaticamente

Da questo punto di vista qui siamo deboli.

Guardando l’intorno ci sono due tipi di mosse, ovvero quelle che inseriscono un task che fanno sempre aumentare il costo ma fanno ridurre il numero di tasks scoperti, se tolgo invece viceversa.

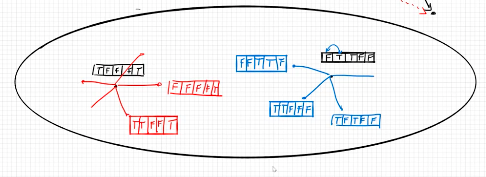
Quello che succede è che ho queste due mosse e non una che va direttamente dall’inizio alla fine:





Altri due possibili intorni sono:

- **SWAP,** o meglio uno swap tra uno che è T e uno che è F



PROBLEMA: ho sempre lo stesso numero di shift, anche se migliora la discesa

--> definito da due posizioni

ADD, DEL (aggiungo uno shift, lo 0, e tolgo lo shift 1)

<0, 1>

Devo stare attento all’enumerazione perché questa è dipendente dallo stato

Immagine che contiene testo

Descrizione generata automaticamenteAd esempio:

Il vicinato che consideriamo mette assieme questi due:   
- **SWAP OR FLIP:** come lo swap con l’aggiunta che uno dei due attributi della mossa può essere anche un valore convenzionale ‘-1’

Immagine che contiene testo

Descrizione generata automaticamente

Questo vuol dire che tolgo 1 ma non aggiungo niente che è come fare la mossa flip su 1.

Quindi:

Immagine che contiene testo, esterni, lavagnabianca

Descrizione generata automaticamente

Ho aggiunto il -1 da entrambe le parti (ho saltato la mossa nulla che non ha senso, ovvero

<-1, -1>, nonché aggiungi niente e togli niente) (mosse in rosso fanno tutte flip)

**COST FUNCTION** = COSTO DI UNA MOSSA dato dallo SHIFT COST (soft, lo posso accettare di pagare), Distance To Feasibility = UNCOVERED TASKS (quando ho uno stato questo potrebbe avere die tasks scoperti – hard, perché devo coprire tutti i tasks)

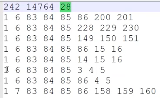
--> Ho quindi due componenti di costo e quindi due DELTA

**INITIAL SOLUTION**: RANDOM

Per fare una soluzione iniziale random di T/F si fanno una serie di random bool:

s[i] = Random (0, 1)

Questa però non è una buona scelta, perché guardando le istanze, le soluzioni sono fatte di tantissimi false (es. 14764) e pochi true (es. 28)



Quindi con random bool non rientro in questa situazione, ma con equilibrio.

Per questo scrivo una soluzione random ma già in linea con la soluzione finale, quindi io assumo di sapere quanti shift servono, guardando mediamente quanti tasks copre uno shift e facendo il numero di tasks diviso questo numero

Quindi ottenuto questo numero = X, do ai vari shift una probabilità pari a X% di avere T, e (1-X) % di avere F

ESEMPIO:

Se ho 1345 shift e ho deciso che me ne servono 30, tiro per ogni shift un numero random compreso tra 0 e 1344, se questo numero che esce è < 30 allora lo prendo, e lo prendo con una probabilità che è 30/1345

**IMPLEMENTAZIONE SU CODICE**

**NEXT**

**Immagine che contiene testo, lavagnabianca, tessuto

Descrizione generata automaticamente**

Questa è la sequenza giusta ma per comodità le genero tutte le mosse:

Immagine che contiene esterni

Descrizione generata automaticamente

E così via

Quindi uso Anynextmove e poi butto via quelle che non servono facendo Feasiblemove.

**RUN CODICE**

Immagine che contiene testo

Descrizione generata automaticamente



Ha messo 5 dei 77 presenti nell’istanza

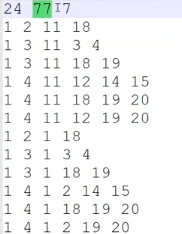


Immagine che contiene testo

Descrizione generata automaticamente

Se io faccio 5 – print all neighbors

Immagine che contiene testo

Descrizione generata automaticamente



Noto che nella colonna di dx, ovvero quelli da togliere compaiono solo 4, 30, 68, 73 e -1 (e non ho questi numeri a sx)

**COSTS**

Immagine che contiene testo

Descrizione generata automaticamente

L’asterisco \* vuol dire che è una componente hard, quindi sono violazioni

**NB: TUTTI I DATI SONO PRIVATI, GESTITI INTERNAMENTE DALLE FUNZIONI INSERT SHIFT E REMOVE SHIFT (NON DEVO MANOVRARE DIRETTAMENTE I DATI MA DEVO AVERE UNA BUONA MODULARITA’ E INFORMATION HIDING)**

13/05/2021

ESEMPIO PASSAGGIO DA RIGA DI COMANDO DI INIT\_SATE



LANCIARE METODO DA RIGA DI COMANDO (alcuni hanno bisogno di alcuni parametri)

--> Chiamata metodo steepest descent (me = abbreviazione, vengono interpretate dal compilatore)

Immagine che contiene testo

Descrizione generata automaticamente

--> hill climbing

Immagine che contiene testo

Descrizione generata automaticamente

--> simuletad annealing (ha bisogno di più parametri)

Immagine che contiene testo

Descrizione generata automaticamente

--> tabu search

HO FILE CON SCRIPT PYTHON PER VALUTAZIONE PARAMETRI SA (FUORI PROGRAMMA)

Fatto per i parametri evidenziati e il 4° viene valutato sula base del fatto che il numero totale di iterazioni è lo stesso



Se lancio lo script ottengo:



19/05/2021

**APPLICAZIONE COMPLESSA COURSE TIMETABLING UNIUD**

- **courses**: #lections, #students, #min number of days per lections, #teacher

- **periods**: couple day x slot

- **rooms**: capienza

- **curricula**

- (hard) **constraints**: devono essere fatte tutte le lezioni indicate, evitare i conflitti tra stessi curricolum o per docente, occupazione delle aule, disponibilità docente

- (soft constraints --> se non si riescono ad ottimizzare pazienza) **objectives**: dimensione delle aule, compattezza = studenti di un certo curriculum devono avere lezioni accorpate – concetto lezione isolata --> è una violazione soft, un corso deve essere spalmato su un certo numero minimo di giorni, per quanto possibile un giorno deve essere fatto in una stessa stanza)

--> *8 termini* di cui:

-- stanno fuori dallo spazio di ricerca (mai violati), semplicemente si fanno una soluzione finale e delle mosse che non violano tutto questo

-- stanno nella funzione di costo (quindi ho 6 componenti di costo) (i due hard hanno peso 1000 perché hard, degli altri 4 ha un peso 1 la room capacity, e altri 3 sono multipli di questo)

--> **Il vettore di questi pesi viene mandato al costruttore**

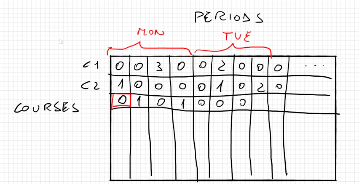
**SEARCH SPACE**

--> matrice in cui ogni riga rappresenta un corso e ogni colonna un periodo, organizzati in giornate, con valore dell’aula (numerate a partire da 1) che occupa quel corso in quel periodo o 0 se non c’è lezione-non è occupata

--> quindi il numero di non 0 per riga deve essere esattamente pari al numero delle lezioni

--> qui il numero di lezioni nella matrice sia pari al numero di lezioni fissato (numero di non 0 è fisso)

--> se c’è una non disponibilità li non ci metto mai in un non 0 (quindi casella rossa avrà sempre uno 0)



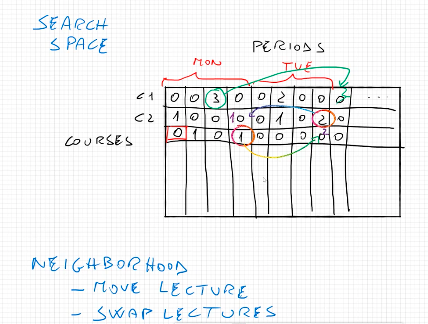
**NEIGHBORHOOD**

- prendo una lezione (non 0) e la sposto in un nuovo periodo in una nuova aula --> MOVE LECTURE (**facciamo questo**)

--> mossa con molti (4) attributi indipendenti:

<corso, lezione all’interno di quel corso che voglio muovere, dove la voglio muovere = nuovo periodo, nuova aula> = <c, l, np, nr>

­- SWAP LECTURE, più facile visto che l’orario è spesso incasinato





In realtà non sono 4 attributi ma 6 e sono i seguenti:

1. Corso (c)
2. Indice lezione (l)
3. Periodo vero lezione (p) --> dato calcolato
4. Nuovo periodo (np)
5. Vecchia aula (r) --> dato calcolato
6. Nuova aula (nr)

Enumerazione enumera i 4 attributi indipendenti

**PROBLEMATICA LEZIONE ISOLATA**

Immagine che contiene testo, lavagnabianca

Descrizione generata automaticamente

Riga diritta = divisione del giorno, quindi una è l’ultima del mercoledì, altra ultima del giovedì.

Queste sono entrambe isolate anche se sono appiccicate, quindi devo andare giorno per giorno per decidere se un qualcosa è isolato o meno.

Quindi una lezione è isolata se ha uno zero a sx e uno a dx quanto sta nel mezzo, se ultima della giornata (3) se ha zero a sx, se prima se ha zero a dx.

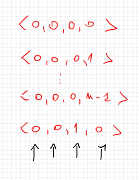
Questo discorso di lezioni isolate va per curriculum cioè per studente.

**NOTA:** ricordo che come mossa posso spostare solo l’aula, solo il periodo o entrambi aula e periodo

20/05/2021

**FIRST E NEXT**

Io ho 4 attributi e faccio una sorta di enumerazione, ma questi attributi sono diversi da loro diversamente da altri problemi trattati, quindi uno 0 è un first

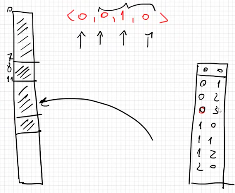


**RANDOM MOVE**

Se io tiro a caso il corso e all’interno del corso tiro a caso la lezione, favorisco le lezioni dei corsi con poche lezioni (probabilità = 1/numero lezioni corso); io voglio che tutte le lezioni abbiano la stessa probabilità di essere smosse.

Quindi posso mettere un vettore che metter tutte le lezioni e poi tiro a caso una fra quelle lezioni in modo da avere delle scelte equiprobabili

O meglio non ho proprio un vettore ma una matrice che mi dice per ogni corso ho una coppia che mi dice corso e lezione (per capire quando la scelgo di quale corso sto parlando).



**PER VALUTARE ERRORI**: scelgo 9 dal menù e mi prova tutte le mosse

Immagine che contiene testo

Descrizione generata automaticamente