Contents

1 Modi per far cagare di meno (trucchetti)

abbiamo visto da prima che la dfs è molto meglio quando si ha una quantità finita di memoria, purtroppo non è proprio completa la bfs è completa ma infattibile

1.1 DFS Patchata a cazzo

come facciamo ad avere tipo la dfs ma non incompleta?

```
def dfs_patchata_a_cazzo(grafo):
    for i in range(1, arrenditi):
        res = dfs_ma_bloccati_a(profondità=i, grafo=grafo)
        if is_good(res):
            return res
```

- la `dfs_patchata_a_cazzo` si chiama *iterative deepening*[1]
- `dfs_ma_bloccati_a` si chiama `DLS` o *Depth Limited Search*

scritta meno alla cazzo di cane

[1] i doppi sensi sono lasciati al lettore

1.1.1 DLS

```
def depth_limited_search(problem, node, levels_left):
    """
            questo ritorna
            1. cutoff: se si è sbattuti alla fine, e FORSE il risultato è
        dopo
            2. failure: mission failed, we'll get 'em next time
            3. un nodo: se si è trovata una soluzione entro la distanza
            """
    cutoff_occured = False
    if problem.is_goal_state(n.state):
        return n
    elif levels_left == 0:
        return "cutoff"
    else:
        for act in problem.actions(n.state):
            c = Node(problem, node, action)
            res = depth_limited_search(problem, node, levels_left - 1)
            if res == "cutoff":
                cutoff_occured = True
            else:
                if res != "failure":
                    return res

    if cutoff_occured:
        return "cutoff"
    else:
        return "failure"
```

1.1.2 Iterated Deepening

iterare con profondità sempre crescenti

```
def iterative_deepening(problem):
    for i in range(1, qualche_limite):
        res = depth_limited_search(problem, Node(problem.initial), i):
        if res != "cutoff":
            return res
```

la chiamata `depth_limited_search(problem, Node(problem.initial), i):` è solo per iniziare una dfs sul problema, ma settando la profondità massima raggiungibile a `i`

1.1.3 Analisi

l'analisi (in tempo e spazio) è lasciata come esercizio (tanto è anche sul libro sticazzi)

lo spazio è lo stesso della dfs

il temp è lo stesso della bfs (in $O - grande$, poi grazialcazzo che questo passa sugli stessi nodi un migliaio di volte quindi un pochino di più ha senso lo faccia)

- in $O - grande$ sarebbe dire che *"l'ultima iterazione è quella che se li mangia tutti"* (che grazialcazzo è esponenziale col branching factor)

in caso di step cost costante trova la soluzione ottima, come la bfs, infatti l'iterative deepening "simula" la bfs

1.2 Ricerca Bidirezionale

Visto che qui si va esponenziale di profondità, se riuscissi ad andare avanti dallo start e indietro dalla soluzione, per "vedere dove si incontrano", farei metà della profondità. La farei due volte, ma sticazzi, ho dimezzato l'esponente per poterla fare, come si può notare dalla descrizione data, bisogna

- conoscere il problema
- poter andare sia avanti che indetro con gli step

lo pseudocodice è dato come esercizio perchè sì

2 A*

`A*` è un figo della madonna, è un'evoluzione di dijkstra, vogliamo migliorare l'efficienza, in tempo e in spazio, di uniform cost.

Se abbiamo una qualche idea di dove sta il goal, esempio google maps, allora espandere la ricerca a palla è abbastanza inefficiente, invece di andare tutto intorno, si vorrebbe quindi muoverci tendendo a espandere verso il goal si espandono per prime delle direzioni più promettenti per fare questo si usa una cosiddeta *euristica*, in questo caso eurisitica vuol dire una *stima* del costo verso il goal, nel caso di google mapsN ad esempio, una distanza a linea d'aria potrebbe essere una stima.

2.1 Euristica

una funzione euristica (heuristic, in inglese, che diocane è il frasconi) $h(n)$ sitma il costo da `n.state` al goal di solito l'euristica si ottiene rilassando il problema in qualche modo

- usare solo `h(n)` come funzione di costo(come `g(n)`) porta a una ricerca *Greedy Best First* (oppure ricerca Greedy e basta)
  - non è ottimo
  - quando riescie però cazzo è veloce, prendi tante decisioni quanti sono gli incroci che incontri
- se si usa `h(n) + g(n)` per fare l'ordinamento della coda[2] ecco che abbiamo `A*`
  - l'ottimalità o meno di `A*` è possibile ma dipende da alcune caratteristiche di `h(n)`

quale potrebbe essere la migliore $h(n)$ possibile? Se avessi come $h(n)$ il costo effettivo del cammino minimo allora amen, andrei sempre verso la soluzione, il branching factor sarebbe virtualmente 1.

2.1.1 Carattersitiche dell'euristica

ammissibile non sovrastima mai il costo

consistente se

$$h(n) \leq costo(n, a, n') + h(n') \ \forall \ n, n', a$$

- nota ai mortali come : se rispetta la disuguaglianza triangolare

si può dimostrare che usare `A*` con un'euristica che preserva queste caratteristiche preserva l'ottimalità

tutte le euristiche consistenti sono ammissibili, ma esistono euristiche ammissibili che non sono consistenti, si possono costruire euristiche ammissibili non consistenti

una cosa bellina per debuggare il codice è fare un fottio di logging (noto anche come debuggare coi print, ma *enterprise*)

la consistenza è complicata da dimostrare, l'ammissibilità è una cazzata da dimostrare

[2] nota anche come `f(n)` [3]

[3] nota anche come funzione di valutazione

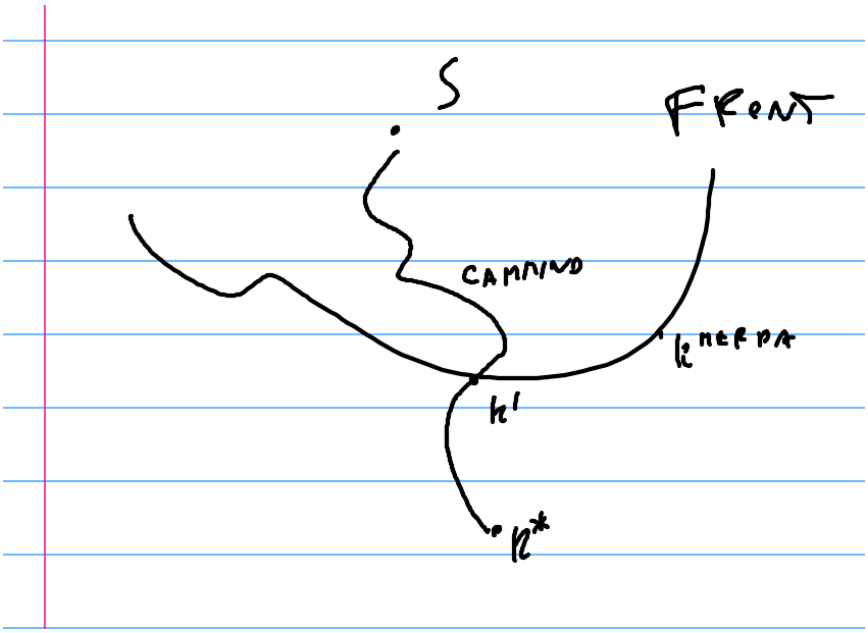
la consistenza vuol dire che la `f(n)` di valutazione non diminiusce mai lungo un cammino de boh, a volte per dire euristica consistente si dice anche euristica *monotona*

- $f(n') := g(n') + h(n')$
- visto che $g(n') = g(n) + c(n, a, n')$
- quindi $= g(n) + c(n, a, n') + h(n')$

2.1.2 Dimostrazione che `A*` resta figo

Teorema `A*` con una euristica consistenza è ottimale

la dimostrazione è facilissima, è lo stesso identico ragionamento di dijkstra qual'era l'ipotesi chiave in dijkstra, era che gli step cost fossero non negativi (o anche solo positivi visto che erano $>= \varepsilon$) qui invece del costo positivo, per sfruttare la ... si usa la *monotonicità*



- siamo in un generico step di `A*` in cui non ho incontrato un goal
- ogni cammino che congiunge uno stato esplorato con uno stato non esplorato incrocia la frontiera

prendiamo un nodo della frontiera, n', che sarà in un cammino da start a goal prendiamo un altro nodo, n^{merda}, nella frontiera, prendiamo per assurdo che viene estratto n^{merda}

visto che $h(n)$ è ammissibile sappiamo che $h(n*) = 0$ visto che non possiamo sovrastimare il costo nullo dal goal al goal

- sappiamo che $f(n') \leq f(n*)$
- sappiamo anche che $f(n*) = g(n*) + h(n*) = g(n*) + 0 = g(n*)$
- $f(n') \leq f(n)$

... siccome è il goal ottimo ... mettete le due insieme e viene la terza (time for libro)

2.1.3 Costruirsi delle euristiche efficaci

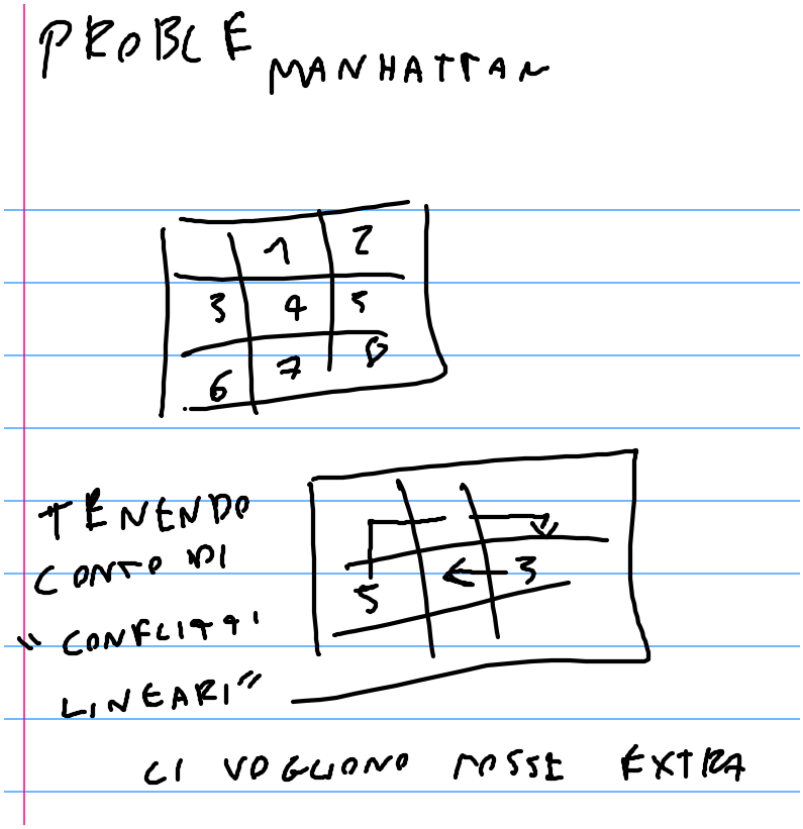
facciamo una successione di euristiche, via via più furbe, per lo SAM MOTHERFUCKING LLOYD

1. Brutale 1 un'idea sarebbe avere come `h(n)` il numero di celle fuori posto del puzzle (escludendo il blank perchè il prof non si ricorda se includere il blank sovrastima o no)

   sappiamo che si fa come minimo una mossa per mettere a posto una cella, quindi usare questo numero come euristica sappiamo che non sovrastima

2. Brutale 2 possiamo usare comunque avere un lower bound per quante mosse ci vogliono per arrivare da una cella al posto della cella, lower bound dato da una distanza di manhattan tra la cella e dove dovrebbe stare la cella, visto che una mossa va sempre di un'unità di sopra o sotto

3. un Po' meno brutale



4. Quale meglio la terza è più informata la terza è più vicina a quanto direbbe l'oracolo perchè è più elevata

   si può dimostrare che usando un'euristica meno informata(meno elevata) si espandono tutti i nodi di un'euristica più informata(più elevata) più altri

   un'euristica più informata va più stretta

   - sfera senza euristica
   - euristica non troppo infromata va schiacciatina
   - euristica più informata va schiacciata
   - oracolo va dritto, infinitamente schiacciato

   quanto guadagno è difficile da calcolare a priori, calcolare la bontà di un'euristica, e scazzare con le euristiche in generale, è una scienza empirica, fatta per sgozzare matematici

(se l'euristica e il costo sono numeri interi, se sono numeri reali so' cazzi a volte)

2.1.4 Euristicone, de novo

effective branching factor si metta che tutti i figli hanno esattamente B figli[4], allora l'albero avrebbe

$$1 + B + B^2 + B^3 + B^4 + B^5 + \cdots + B^d = \frac{B(B^d - 1)}{B - 1}$$

figli sapendo quanti nodi ho espanso, e quale profondità ho raggiunto, posso ricavare la B che mi farebbe tornare tutto, e usarla come "valore atteso"

penetranza ::[5] si calcola come

$$penetrance = \frac{d}{N}$$

[4] "seventh son of a seventh son" momento

[5] le battutine del cazzo[6] sono lasciate al lettore

[6] o sul cazzo