Università degli studi di Catania
Dipartimento di Matematica e Informatica
Corso di Laurea Magistrale in Informatica
A.A. 2019/2020

CAPACITATED VEHICLE ROUTING PROBLEM

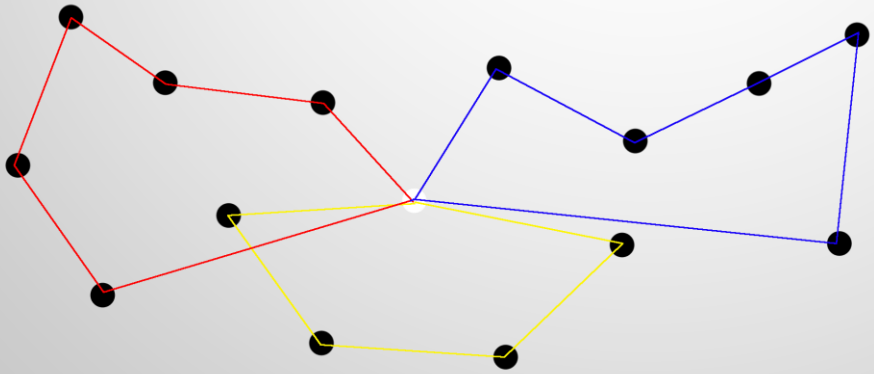
WITH A HYBRID IMMUNOLOGICAL ALGORITHM

Antonio Ganci
W82000167

DEFINIZIONE DEL PROBLEMA

Il Capacitated Vehicle Routing Problem (CVRP) è uno dei classici problemi di ottimizzazione nell'ambito della ricerca operativa.

Consiste nella gestione di un insieme di veicoli con capacità di carico limitata che, partendo da un deposito comune, devono essere distribuiti su un certo numero di città al fine di determinare una serie di percorsi di costo minimo per ciascun veicolo in modo da soddisfare un dato insieme di clienti.



FORMALMENTE…

Il CVRP è formalmente definito come un grafo non orientato $G = (V, E)$ dove V è l'insieme dei vertici ed E è l'insieme degli archi. Il vertice v_0 corrisponde al deposito, dal quale partono k veicoli con capacità di carico C. Ad ogni vertice è associata una richiesta q_i e ad ogni arco e_{ij} un valore non negativo che rappresenta il costo del viaggio tra la città i e la città j.

L'**obiettivo** del CVRP è quello di determinare un insieme di *k-routes* di **costo minimo**, tale che:

1. Ogni percorso inizia e finisce al *deposito;*
2. Il carico totale delle richieste q_i su ogni *route* non deve eccedere la capacità C;
3. Ogni città deve essere attraversata solamente una volta e da un solo veicolo.

ALGORITMO IMMUNOLOGICO

Un algoritmo immunologico è un algoritmo bio-ispirato basato su clonal selection principle che cerca di emulare il comportamento del sistema immunitario naturale.

Si basa su due elementi principali, gli **antigeni**, che rappresentano il problema da affrontare e le **B-cells** che rappresentano un punto nello spazio di ricerca.

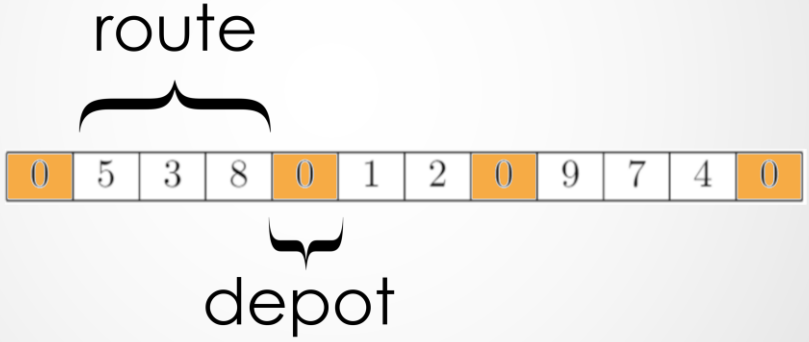
L'algoritmo si sviluppa in 5 fasi principali:

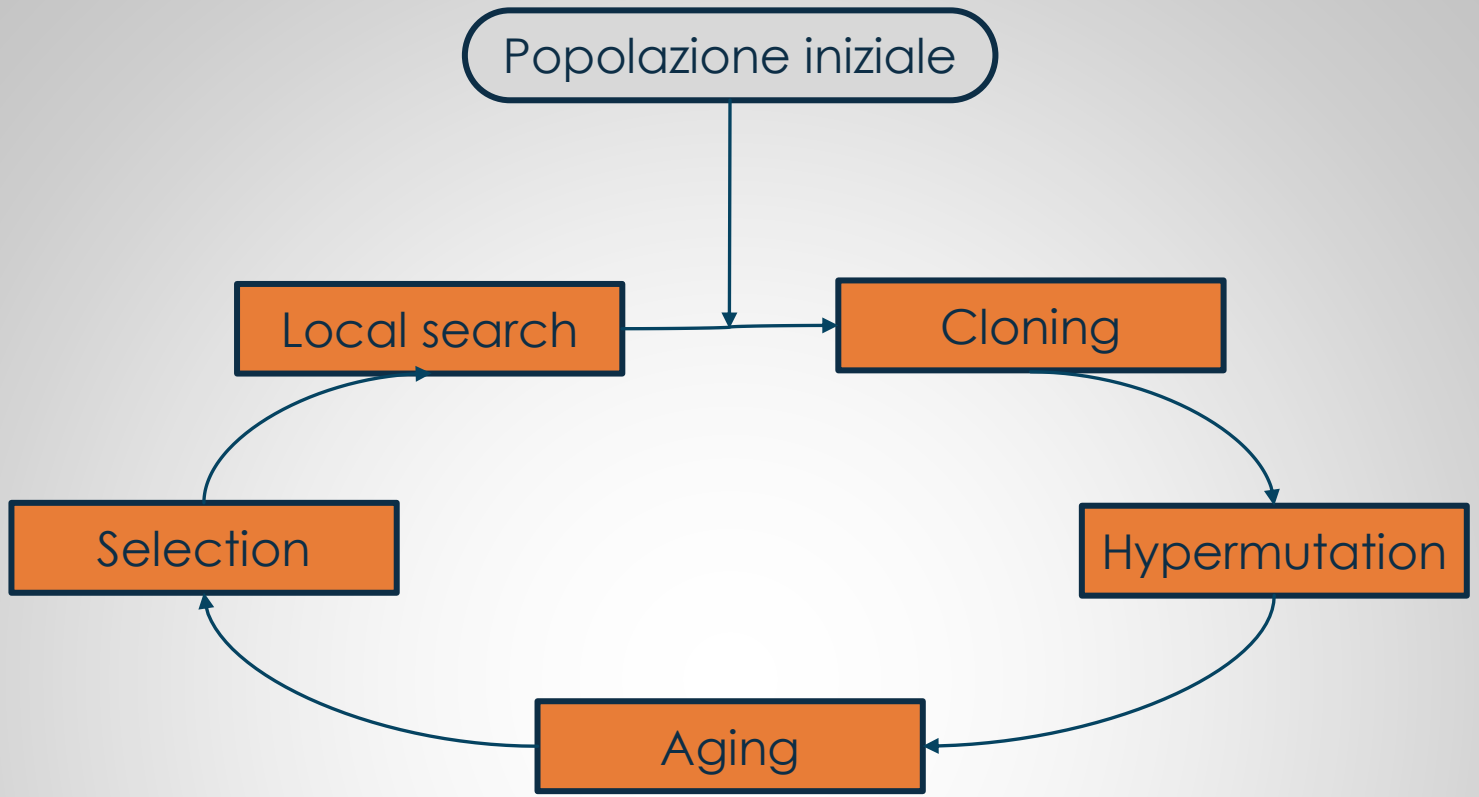
1. Cloning;
2. Inversely Hypermutation;
3. Aging;
4. Selection;
5. Local Search.

APPLICAZIONE AL CVRP

Ogni soluzione o B-cell è rappresentata come un array di interi corrispondenti agli id dei nodi (clienti/città). L'array contiene k+1 nodi deposito, dove k è il numero di veicoli dell'istanza. I nodi giacenti tra un deposito e un altro rappresentano una route.

Al momento dell'inizializzazione, alla B-cell vengono assegnati un'**età** casuale tra e una **fitness** calcolata sommando le distanze tra i nodi di ogni route.





IMPLEMENTAZIONE

INVERSELY HYPERMUTATION

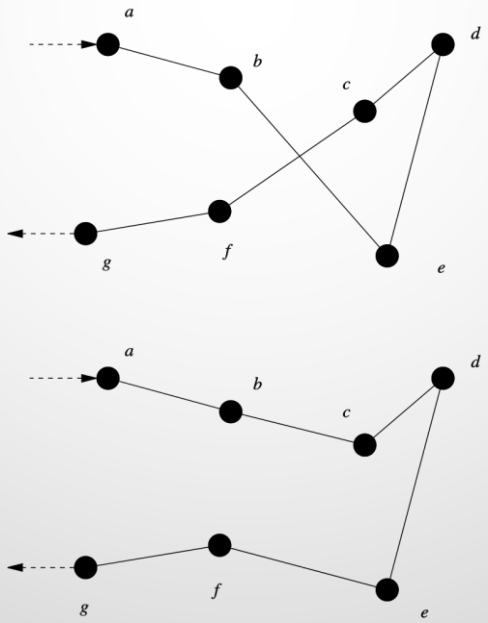
Ogni soluzione viene mutata un numero di volte inversamente proporzionale alla propria fitness.

La mutazione consiste nell'applicazione casuale di uno dei seguenti operatori:

- **Move:** sceglie un nodo casuale e lo inserisce in una posizione, anch'essa casuale e shifta tutto il resto;
- **Reverse:** sceglie un intervallo casuale e inverte l'ordine dei nodi al suo interno.

LOCAL SEARCH
TWO-OPT ALGORITHM

In questa implementazione è stato aggiunto un operatore di local search noto come **two-opt**. Questo operatore cerca di raffinare ancora di più le B-cell selezionate per la nuova generazione.
L'idea alla base del **two-opt** è quella di cercare un percorso in cui due o più archi si incrociano e riordinarlo in modo che questo non accada più.



TESTING

Per il testing dell'algoritmo sono state utilizzate 5 istanze standard del CVRP di complessità crescente. Per ogni istanza sono state realizzate 5 runs di 10000 epoche ciascuna. Per ogni run si è mantenuta una popolazione di 100 elementi con un'età massima di 5 epoche, un fattore di duplicazione pari a 2 e un mutation rate di 10.

Di seguito i risultati dell'algoritmo su ogni istanza.

Instance	Best	Worst	Mean	Std. Dev.	Best Known
A-n32-k5	**784**	784	784	0	**784**
A-n39-k6	**831**	847	840	7,15	**831**
A-n45-k7	**1146**	1211	1165,2	26,3	**1146**
A-n62-k8	1603	1321	1313,6	8,23	1290
A-n80-k10	1817	1855	1842	16	1764

CONCLUSIONI

I risultati mostrano che l'efficacia dell'algoritmo varia in base alla complessità dell'istanza del problema. In particolare, l'algoritmo riesce a trovare abbastanza facilmente la soluzione migliore in istanze del problema relativamente piccole mentre non riesce a farlo nelle istanze più grandi pur avvicinandosi parecchio.

Potrebbe essere interessante testare l'algoritmo utilizzando qualche altro tipo di operatore di hypermutation o magari combinarne di esistenti.

L'algoritmo può essere indubbiamente migliorato soprattutto per quanto riguarda le prestazioni computazionali magari cercando di parallelizzare il codice dove possibile e/o utilizzare un linguaggio di programmazione più efficiente.