
UNIVERSITÀ DEGLI STUDI DI CATANIA

DIPARTIMENTO DI MATEMATICA E INFORMATICA

CORSO DI LAUREA MAGISTRALE IN INFORMATICA

---

# UNCAPACITATED FACILITY LOCATION PROBLEM

# WITH

# IMMUNOLOGICAL ALGORITHM

---

PROGETTO DI COMPUTAZIONE NATURALE

---

Andrea Sequenzia
Matricola: W82000160

---

ANNO ACCADEMICO 2018/2019

# Indice

# 1 Introduzione

Una gestione efficiente della catena di approvvigionamento porta ad un ovvio aumento del profitto, ad un aumento della quota di mercato, alla riduzione dei costi operativi, e si riesce a soddisfare in modo migliore il cliente. Una delle decisioni strategiche che si possono compiere è quella di scegliere la posizione delle strutture (facility). Per compiere queste decisioni nascono i **Location problem** ed essi sono classificati con alcune assunzioni come limiti delle capacità o numero di siti aperti [4]. In questa relazione viene considerato il problema **Uncapacited Facility Location**, uno dei problemi più studiati nell'ottimizzazione combinatoria. In esso si assume che il costo per soddisfare i requisiti del cliente ha due componenti: un costo fisso per installare una struttura su una determinata location e un costo di trasporto per soddisfare il cliente dalla struttura. In letteratura è stato trattato con diversi nomi: uncapacitated, simple, optimal seguito da plant, warehouse, facility, site, seguito da location [5]. È stato dimostrato che UFLP è un problema NP-Hard [3]. UFLP è stato trattato in vari modi, in questo progetto verrà utilizzato un algoritmo immunologico, ovvero un algoritmo basato su sistema immunitario naturale. Il suo funzionamento verrà approfondito nella terza sezione di questa relazione.

# 2 Definizione del problema

In un problema UFLP, ci sono $m$ clienti da servire ed $n$ possibili location dove poter stabilire delle facilities che servano i clienti e soddisfare la domanda. Ogni facility $i$ ha un costo fisso di installazione $f_i$. Un costo di trasporto $c_{ij}$ per servire il il cliente $j$ dalla facility $i$. Le facility non hanno limiti nelle capacità di quanto possono servire e non ci devono essere clienti non soddisfatti. L'obiettivo finale è quello di stabilire quali facility attivare, soddisfando tutti i clienti, minimizzando il costo per ottenere ciò. Formalmente:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{i=1}^{n} f_i y_i \\
\text{subject to} \quad & \sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, ..., m, \\
& x_{ij} \leq y_i, \quad i = 1, ..., n, j = 1, ..., m, \\
& x_{ij}, y_i = 0, 1, \quad i = 1, ..., n, j = 1, ..., m
\end{aligned}
$$

dove

$c_{ij} =$ il costo per soddisfare la domanda del cliente $j$ dalla facility $i$;

$f_i =$ il costo per stabilire la facility nella location $i$

$$
x_{ij} = \begin{cases} 1 & \text{se il cliente j viene servito dalla facility i,} \\ 0 & altrimenti; \end{cases}
$$

$$
y_i = \begin{cases} 1 & \text{se la facility è stabilita nella location i,} \\ 0 & altrimenti; \end{cases}
$$

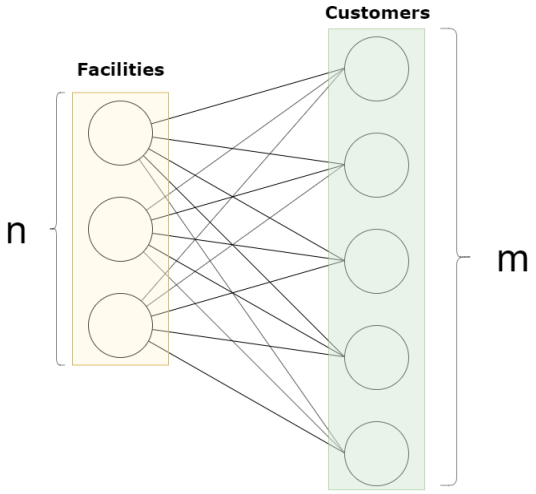


Figura 1: Esempio problema UFLP

Il problema può essere matematicamente decomposto in due sottoproblemi indipendenti [1]:

1. **Location**, ovvero le facilities da stabile ($y_i$)
2. **Allocation**, per le facility stabilite, determinare la distribuzione e quindi quali archi usare per ogni customer ($x_{ij}$)

Per ogni soluzione del problema di Location, una soluzione ottima può essere ottenuta per il problema Allocation. Dato un vettore $y$, può essere ottenuto un assegnamento ottimale delle $x_{ij}$ usando la seguente formula:

$$k : \min_k c_{kj}, \quad k = 1, ..., n$$

$$x_{ij} = \begin{cases} 1 & \text{se i=k,} \\ 0 & altrimenti; \end{cases}$$

Quindi il problema si riduce a determinare l'assegnamento di facilities ottimale.

# 3 Algoritmo immunologico

Un algoritmo immunologico è un algoritmo basato sul clonal selection principle, esso è un ottimo esempio di strategia intelligente in cui l'adattamento opera a livello locale, mentre un comportamento complesso agisce a livello globale. L'idea di questa tipologia di algoritmo deriva dal sistema immunitario naturale, in particolare da come le cellule si adattano per legarsi ed eliminare entità straniere, meglio conosciute come Antigene. Questo algoritmo è basato su due entità: gli **antigeni** che rappresentano il problema da affrontare e le **B cell** (Linfocita B) che rappresentano un punto nello spazio di ricerca [6]. L'algoritmo crea una popolazione di B cell, esse cresceranno e si perfezioneranno nel corso di un numero prefissato di generazioni, morendo quando hanno raggiunto un'età prefissata. La qualità delle B cell si valuta attraverso una funzione di fitness. Come primo passo, l'algoritmo genera una popolazione random $P^{(t)}$ e successivamente usa i seguenti operatori in sequenza:

1. **Cloning**: questo operatore simula il meccanismo di proliferazione di un sistema immunitario. Esso genera una popolazione intermedia $P^{(clo)}$ copiando *dup* volte ogni B cell, assegnando un'età casuale tra $[0, \frac{2}{3}\tau_b]$, la nuova popolazione così fatta sarà delle dimensioni $d \times dup$.

2. **Inversely Hypermutation**: il compito di questo operatore è quello di esplorare il vicinato di ogni clone creato dal precedente operatore. Ciò viene compiuto eseguendo $M$ mutazioni ad ogni elemento di $P^{(clo)}$ e ottenendo la nuova popolazione $P^{(hyp)}$. Per determinare $M$, viene adottata una legge inversamente proporzionale al valore della fitness: più piccolo (minimizzazione) è il valore valore della funzione di fitness, meno mutazioni saranno fatte alla B cell. Dato un clone $x$, per determinare il valore $M$ è necessaria la seguente formula:

$$\alpha = e^{-\rho \hat{f}(x)} \tag{1}$$

   dove $\alpha$ rappresenta il mutation rate,$\rho$ lo shape del mutation rate e $\hat{f}(x)$ è il valore della funzione di fitness normalizzato tra $[0, 1]$. Il numero di mutazioni $M$ è dato da:

$$M = \lfloor (\alpha \times l) + 1 \rfloor \tag{2}$$

   dove $l$ è la lunghezza di una B cell.

3. **Aging**: questo operatore aiuta l'algoritmo a saltare via dagli ottimi locali. Per ottenere ciò, rimuove le vecchie B cell dalla popolazione iniziale e da quella ipermutata. L'eliminazione avviene quando l'età (che viene incrementata ad ogni fine generazione) raggiunge l'età massima ($\tau_B$). Così facendo, si produce alta diversità e si aiuta l'algoritmo ad evitare le convergenze premature. Viene anche fatta un eccezione nella rimozione, ovvero lasciare la migliore soluzione nella popolazione anche se ha raggiunto la massima età. Questa variante viene chiamata *elitist aging operator*.

4. **($\mu + \lambda$)-Selection**: l'ultimo operatore sceglie le migliori *d* B cell da mandare alla generazione successiva. In questo caso si ha $\mu = d$ e $\lambda = (d \times dup)$, questo operatore riduce

la grandezza della popolazione di grandezza $\lambda \geq \mu$ in nuova popolazione di dimensione $\mu = d$. A volte può accadere che siano sopravvissuti meno B cell di quelli richiesti per avere la giusta dimensione della popolazione. In questo caso, vengono generate casualmente delle nuove B cell per raggiungere la giusta dimensione.

**Algorithm 1** Immunological Algorithm (d, dup, $\rho$, $\tau_B, n\_gen$)

$t \leftarrow 0$

$P^{(t)} \leftarrow Initialize\_Population(d)$;

$Compute\_Fitness(P^{(t)})$;

**repeat**

  $P^{(clo)} \leftarrow Cloning(P^{(t)}, dup)$;

  $P^{(hyp)} \leftarrow Hypermutation(P^{(clo)}, \rho)$;

  $Compute\_Fitness(P^{(hyp)})$

  $(P_a^{(t)}, P_a^{(hyp)}) \leftarrow Aging(P^{(t)}, P^{(hyp)}, \tau_B)$

  $P^{(select)} \leftarrow (\mu + \lambda) - Selection(P_a^{(t)}, P_a^{(hyp)})$;

  $Compute\_Fitness(P^{(t+1)})$;

  $t \leftarrow t + 1$;

**until** number of generation reached

# 4 Implementazione

Per implementare questo algoritmo immunologico e adattarlo al problema UFLP è stato usato il linguaggio di programmazione di Python e per le operazioni sui grafi, è stata usata la libreria **networkx**[10]. Per rappresentare ogni soluzione (B cell) sono stati usati dei vettori binari, dove 0 indica che la facility è disattivata e 1 la facility è attivata. La funzione di fitness viene calcolata tenendo conto della configurazione del vettore delle facilities, per ogni customer viene consi-

[10]https://networkx.github.io/

derato l'arco (disponibile) con il minore costo così da soddisfare i requisiti del sottoproblema Allocation. Gli script creati sono i seguenti:

- **immune_algorithm.py**: tramite questo script viene eseguito tutto il flusso dell'algoritmo, quindi vengono implementati i vari operatori e gestita la popolazione;
- **instance_generator.py**: questo script genera un'istanza dal problema da un file di testo che segue un determinato formato dati;
- **solution.py**: questa classe implementa una B cell, in essa è contenuta una permutazione (vettore binario), l'età, la fitness e il metodo per calcolarla. Inoltre è implementato il metodo per eseguire l'ipermutazione;
- **main.py**: questo è lo script da eseguire per eseguire l'intero programma, da esso si possono settare i vari parametri: numero delle generazioni da eseguire, numero di B cell della popolazione, numero di clonazioni, il parametro $\rho$ per l'hypermutation e $\tau_B$;

# 5 Esperimenti e risultati

Per testare l'algoritmo sono state usate delle istanze standard del problema usate nella letteratura contenute nella *OR Library* [2]. Sono stati presi 3 tipi di istanza di diversa difficoltà. Rispettivamente, con m si indica il numero di potenziali location per facility e con n il numero di customer:

1. **cap71**: $m = 16, n = 50$
2. **cap101**: $m = 25, n = 50$
3. **cap131**: $m = 50, n = 50$

Nella Tabella 1 vengono mostrati i parametri utilizzati per ogni istanza, per la prima istanza sono state provate 3 set differenti per confrontare la differenza nei risultati. L'algoritmo è stato

| | **cap71 (set 01)** | **cap71 (set 02)** | **cap71 (set 03)** | **cap101** | **cap131** |
|---|---|---|---|---|---|
| **Generation** | 250 | 300 | 500 | 2500 | 2500 |
| **Population** | 100 | 50 | 100 | 100 | 50 |
| **Clonation** | 3 | 2 | 2 | 2 | 2 |
| $\rho$ | 0.5 | 0.7 | 1 | 0.7 | 0.5 |
| $\tau_B$ | 20 | 25 | 50 | 100 | 100 |

Tabella 1: Parametri usati nelle varie istanze

eseguito 20 volte per ogni istanza e set di parametri, calcolando la media, la deviazione standard, la migliore e la peggiore soluzione trovata. Come si può notare in Tabella 2, gli ottimi

| | **Mean** | **Std. Dev.** | **Best** | **Worst** | **Optimal**[1] |
|---|---|---|---|---|---|
| **cap71(set01)** | 936127.978 | 1774.462 | 933568.89 | 939177.5125 | 932615.750 |
| **cap71(set02)** | 934590.219 | 1298.40 | 932615.750 | 937364.399 | 932615.750 |
| **cap71(set03)** | 933577.401 | 804.237 | 932615.750 | 934829.44 | 932615.750 |
| **cap101** | 803460.405 | 1255.260 | 801420.275 | 804950.45 | 796648.437 |
| **cap131** | 851779.987 | 4313.829 | 845087.975 | 857130.762 | 793439.562 |

Tabella 2: Risultati esperimenti

sono stati trovati solo con due set nella prima istanza (cap71) e nelle altre due istanze i valori erano abbastanza vicini all'ottimo globale. All'aumentare delle generazioni, si aumenta la qualità della soluzione ma ovviamente la durata dell'esecuzione sarà maggiore. L'incremento della popolazione o del fattore di clonazione influisce molto sulla velocità della prestazione dell'algoritmo e non sempre migliora i risultati.

[1] Ottimo globale dato dalla OR Library

# 6 Conclusioni

In questa relazione è stato trattato il problema Uncapacitated Facility Location attraverso un algoritmo immunologico. L'algoritmo può essere migliorato applicando delle euristiche alla generazione della popolazione iniziale come la *net benefit heuristic* proposta in [1]. Lo stesso codice può essere di gran lunga migliorato, implementando il multithreading in processi parallelizzabili come il calcolo dell'allocation o nell'applicazione dello delle mutazioni alla classe. L'aumento di prestazione renderà più veloce compiere più esperimenti con varie configurazioni di parametri, in modo da trovare le configurazioni più ottimali.

# Riferimenti bibliografici


[1] K.S. Al-Sultan and M.A. Al-Fawzan. A tabu search approach to the uncapacitated facility location problem. *Annals of Operations Research*, 86(0):91–103, Jan 1999.

[2] J. E. Beasley. Or-library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, Nov 1990.

[3] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[4] Ali Guner and Mehmet Şevkli. A discrete particle swarm optimization algorithm for uncapacitated facility location problem. *Journal of Artificial Evolution and Applications*, 2008, 04 2008.

[5] Jakob Krarup and Peter Mark Pruzan. The simple plant location problem: Survey and synthesis. *European Journal of Operational Research*, 12(1):36 – 81, 1983.

[6] Mario Pavone and Giuseppe Nicosia. Clonal selection - an immunological algorithm for global optimization over continuous spaces. *Journal of Global Optimization*, 53:1–40, 08 2011.