Corso di Laurea Magistrale in Ingegneria Elettronica e Telecomunicazioni per l'Energia

Alma Mater Studiorum Università di Bologna
Campus di Cesena

A.A. 2021-2022

Mercuriali Gianluca e Tosi Francesco
matr. **0001056381** matr. **0001047560**

data **27/06/2022**

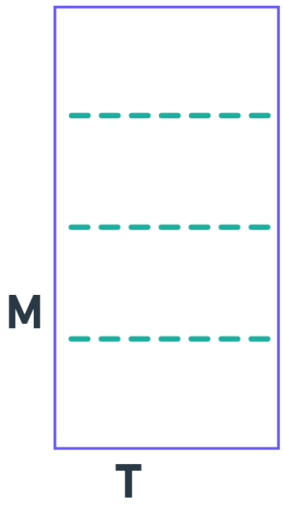
MATEMATICA APPLICATA LM

Gruppo 14**: process allocation**

PROBLEMA: Viene dato un insieme di processi, ciascuno dei quali richiede un tempo di elaborazione e occupa una certa memoria e il cui "profitto" è dato dal prodotto dei due.
Tali processi devo essere elaborati da un processore avente memoria totale M e un tempo massimo di calcolo T per ogni slot.
I singoli processi non si devono interrompere e non sono sovrapponibili e possono essere eseguiti in parallelo affinché per ogni istante di tempo non si superi la memoria massima.

OBIETTIVO: Definire un sottoinsieme di processi, in modo tale da ottenere il profitto massimo possibile.
Per il raggiungimento dell'obiettivo sono stati svolti tre algoritmi: Greedy, Local Search e GRASP.

IMPLEMENTAZIONE: Il problema può essere semplificato come un Bin Packing 2D, con la variante che il bin è unico.
Per questo motivo il bin è stato suddiviso in più "scaffali", ognuno dei quali ha larghezza massima T e altezza M/num.Scaffali come mostrato nella figura seguente:



Nel caso in figura, il numero degli scaffali è quattro

Il numero di processi da inserire all'interno del bin è definito a priori ed è contenuto nella variabile Np, tali processi sono stati generati come dei rettangoli aventi base t e altezza m. I valori sono stati scelti in modo randomico, in particolare per la base i valori generati sono compresi tra 1 e il 20% del tempo massimo T, per i valori di memoria invece la generazione oscilla tra i e il 10% della memoria totale M.
I processi così realizzati in alcuni casi possono risultare quadrati, inoltre è stato scelto di partire da 1 nell'intervallo dei valori perché così non si generano profitti nulli e quindi processi da escludere a prescindere.

GREEDY:
La sua implementazione si trova all'interno della funzione "greedy", inizialmente i processi vengono ordinati in modo decrescente rispetto ai profitti e successivamente vengono inseriti nel bin a partire dal processo avente profitto maggiore.
Il riempimento degli scaffali parte dal basso e si passa allo scaffale successivo quando viene raggiunto il valore massimo di tempo o memoria massima del cassetto.
Se l'i-esimo processo che deve essere inserito supera il tempo massimo o la memoria massima dello scaffale corrente, allora se ne apre uno nuovo che a sua volta verrà riempito seguendo il criterio precedente.
I processi che rimangono fuori dal bin vengono inseriti all'interno del vettore scarto, che successivamente verrà utilizzato per fare gli scambi durante la Local Search.
Il numero di scaffali che possono essere riempiti è definito a priori.
Il bin viene trattato come una matrice che ha numero di righe pari al numero degli scaffali e numero di colonne pari al numero dei processi.
Per tenere traccia della memoria e dei tempi che vengono messi all'interno del bin, vengono realizzate due matrici per tempo e memoria, il cui prodotto genera il profitto di ciascun processo allocato all'interno della soluzione temporanea.

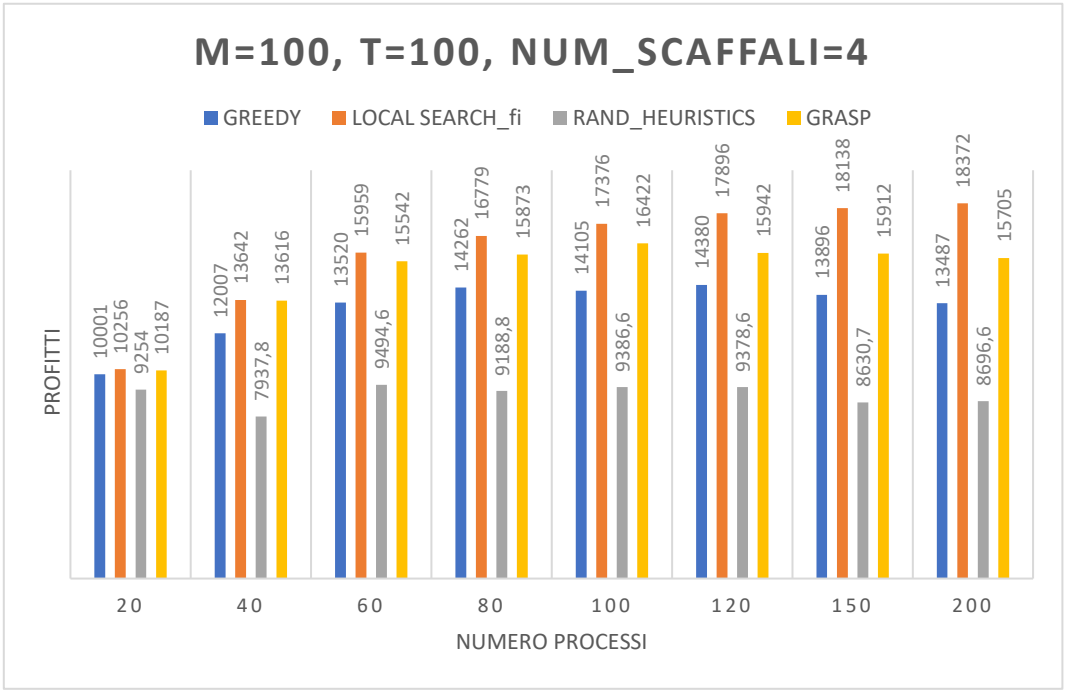
LOCAL SEARCH:
La sua implementazione si trova all'interno della funzione "localSearch_fi", ed è stato scelto l'algoritmo di first improvement.
Si parte dalla soluzione ottenuta dall'algoritmo greedy e si esegue il primo scambio migliorante che si trova all'interno dell'array degli scarti.
Il primo processo che si confronta con il vettore degli scarti è quello inserito per primo all'interno dello scaffale più in basso.
Finché non si trova uno scambio migliorante si continua a scorrere gli elementi contenuti nei cassetti, non appena si trova un profitto migliore si controlla che se scambiato questo non superi il massimo del tempo e il massimo della memoria del cassetto corrente.
Se queste condizioni sono rispettate allora si procede con lo scambio e si aggiorna la soluzione che risulterà avere profitto totale maggiore.
Si procede in questo modo finché non si confrontano singolarmente tutti i profitti in soluzione con tutti i profitti contenuti nello scarto.

GRASP:
L'algoritmo GRASP è realizzato combinando un algoritmo di ricerca locale e un'euristica randomizzata.
L'euristica randomizzata viene implementata all'interno della funzione "randHeuristics", che risulta essere simile alla funzione "greedy" ad eccezione che seleziona in modo randomico i processi da stanziare all'interno del bin.
Il riempimento come negli algoritmi precedenti è svolto partendo dallo scaffale più basso.
Anche in questo caso gli elementi non inseriti in soluzione vengono inseriti all'interno dell'array "scarti".
A partire dalla soluzione ottenuta in modo randomico, si esegue una Local Search per più iterazioni, questo algoritmo è stanziato nella funzione "localSearch_grasp".

Questo algoritmo è identico alla Local Search precedente ovvero che confronta singolarmente tutti i profitti contenuti nella soluzione con tutti i profitti all'interno dello scarto, se viene trovato uno scambio migliorante e rispetta le condizioni di non saturazione del tempo massimo e della memoria dello scaffale corrente, allora lo sostituisce e aggiorna la soluzione.

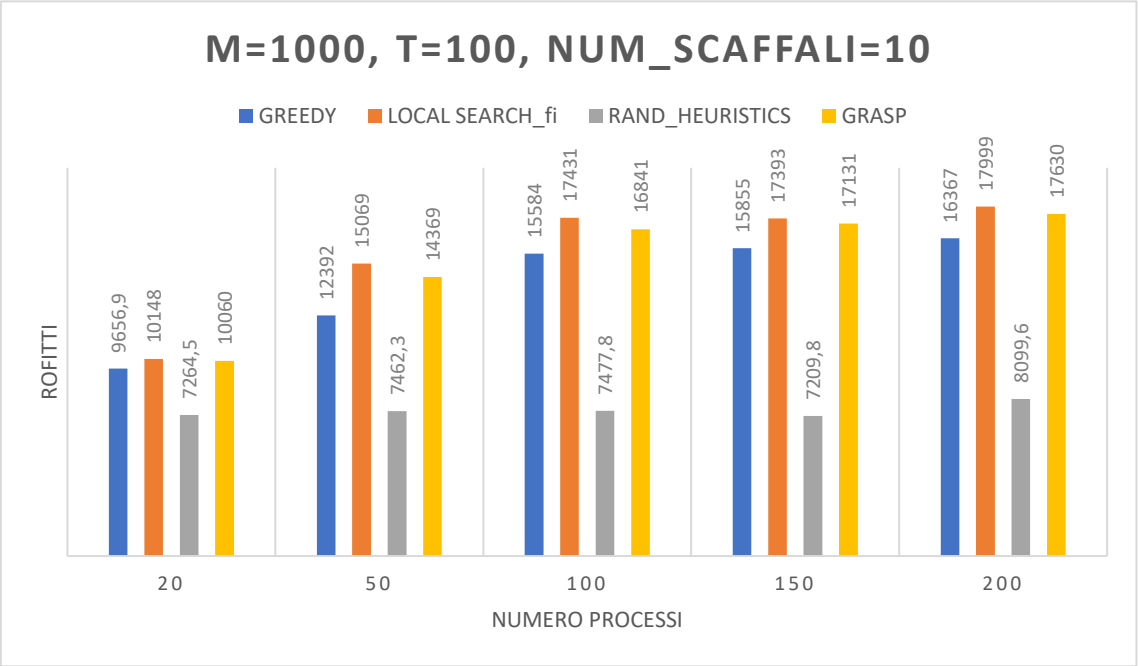
CONCLUSIONI:
Per ottenere risultati più veritieri in output, vengono generati dieci problemi test, i quali danno luogo a dieci soluzioni che vengono memorizzati in un array di cui successivamente si fa la media.

Una prima valutazione dei risultati è stata realizzata con i seguenti parametri: T=100, M=1000, n_scaffale=4 e i processi generati possono avere al massimo il 20% di T e il 10% di M.
Sono stati valutati i vari profitti degli algoritmi al variare del numero di processi generati, come mostrato nella figura seguente:



Si può notare che per ogni valore del numero dei processi la Local Search e il GRASP, migliorano rispettivamente il profitto del Greedy e dell'euristica randomica.
Confrontando ora il profitto del greedy e quello dell'euristica randomica è evidente che il primo algoritmo avrà sempre profitto maggiore in media, perché ordina in mod decrescente il valore dei processi prima di inserirli in soluzione invece che prenderli randomicamente.
Per quanto riguarda il profitto della Local Search, questo è in media sempre maggiore di quello del GRASP, perché parte da una soluzione migliore.
Sempre per quanto riguarda questi due algoritmi presi in considerazione, a partire da Np=100 fino ad arrivare a Np=200, il GRASP non migliora il profitto, anzi lo peggiora a differenza della Local Search che lo migliora sempre e comunque.
Per quanto riguarda i tempi di esecuzione degli algoritmi, il greedy e l'euristica randomizzata hanno tempi equiparabili anche all'aumentare di Np e rimangono pressoché gli stessi, il greedy impiega qualche decimo di millesimo di secondo in più perché riordina in modo decrescente i processi.
Invece considerando gli algoritmi di Local Search e GRASP, i tempi crescono all'aumentare del numero dei processi, perché confrontano maggiori possibili scambi e il GRASP è decisamente più lento perché esegue la Local Search più volte.

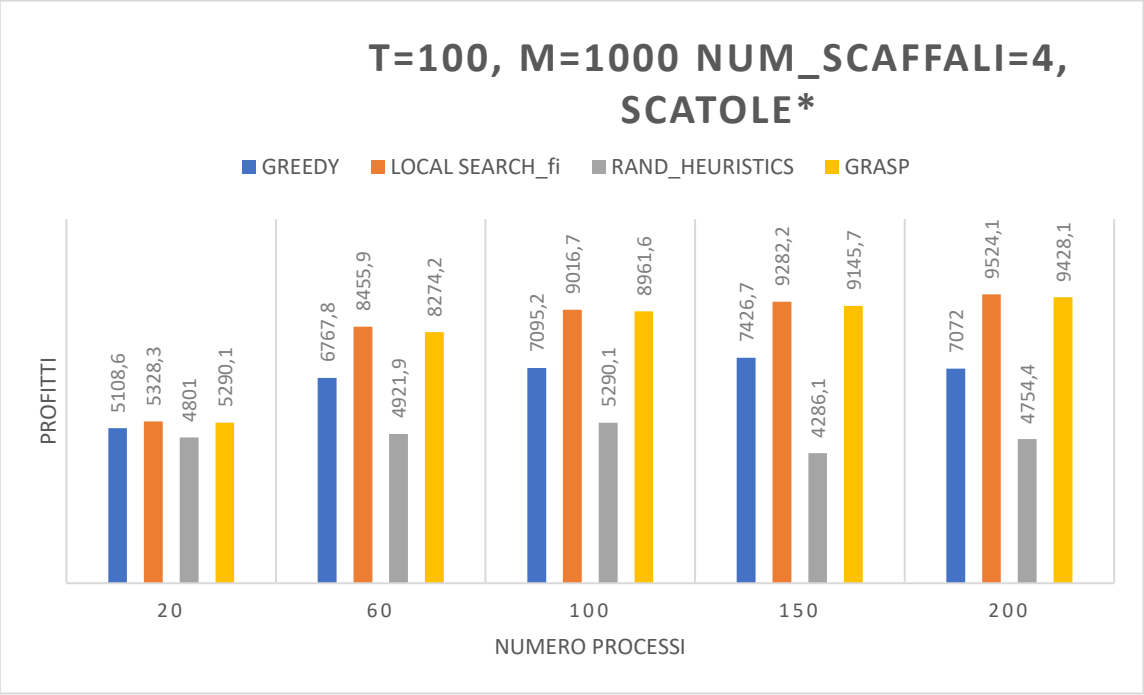
Si è deciso di valutare i risultati a partire dai precedenti parametri, ma aumentando il numero degli scaffali a 10.
Sono stati valutati i vari profitti degli algoritmi al variare del numero di processi generati, come mostrato nella figura seguente:



In questa situazione, con un numero maggiore di scaffali posso allocare più processi in parallelo, ma ciascun scaffale ha memoria massima minore.
Rispetto ai risultati ottenuti in precedenza, avere un numero maggiore di scaffali, mi fa ottenere un profitto del GRASP per valori di Np pari a 150 e 200 che è migliore del 13%, si passa da un profitto con quattro scaffali pari a 15705 ad un profitto pari a 17630 con dieci scaffali a Np=200.
Per quanto riguarda la Local Search invece i profitti sono leggermente inferiori per ogni valore di Np considerato.

Infine, si è voluto valutare i risultati a partire dai parametri iniziali ovvero: T=100, M=1000, n_scaffale=4, ma i processi generati risultano essere più piccoli, perché possono avere al massimo il 10% di T e il 10% di M.

Sono stati valutati i vari profitti degli algoritmi al variare del numero di processi generati, come mostrato nella figura seguente:



In questa situazione, si generano processi che per quanto riguarda il tempo, risultano essere al massimo la metà rispetto al caso precedente, di conseguenza i profitti risultano essere minori per ogni algoritmo a tutti gli Np considerati.