Ricerca Operativa
Homework 2

Maddalena Ghiotti, Aldo Sambo

July 2022

1 Introduzione

L'obiettivo del problema è minimizzare la lateness massima L_{max} su macchina singola, con tempi di setup dipendenti dalla sequenza.
Per ovviare al problema, abbiamo ideato tre euristiche costruttive e tre euristiche iterative. Le prime torneranno utili per generare velocemente una soluzione. Sono proposte in seguito anche euristiche miste, generate dalla combinazione di un'euristica costruttiva con una o più euristiche iterative.
Allo scopo di stabilire il miglior compromesso efficacia-efficienza, gli algoritmi vengono infine confrontati.
Denotiamo l'insieme ordinato dei job con il termine *sequenza*.

2 Euristiche costruttive

Si propongono di seguito i tre diversi criteri di scelta per determinare velocemente un possibile ordine dei job nella sequenza.

2.1 Ordinamento per date di consegna: EDD

Dato un insieme di job e l'insieme delle loro date di consegna, disponiamo i job nella sequenza secondo l'ordine crescente delle loro date di consegna. Ossia

- Al primo posto nella sequenza c'è il job con consegna più urgente.
- All'ultimo posto nella sequenza c'è il job con consegna meno urgente.

Scegliamo come soluzione la sequenza così generata e calcoliamo L_{max} a partire da tale ordinamento.
Su Matlab si utilizza la funzione $sort(...)$ applicata al vettore delle date di consegna e poi si sceglie il vettore degli indici "ordinati" restituito da tale funzione come ordine di esecuzione dei job.

2.2 Ordinamento per tempi di setup: SetUp

Dato un insieme di job e l'insieme dei loro tempi di setup, disponiamo i job nella sequenza, a partire dalla prima posizione, rispettando il seguente criterio:

- Al primo posto nella sequenza poniamo il job con tempo di setup iniziale (rispetto, quindi, alla macchina vuota) più basso tra quello di tutti i job.
- All'i-esimo posto nella sequenza poniamo il job con tempo di setup, rispetto al job nella posizione subito precedente, più basso tra quello di tutti i job non ancora inseriti nella sequenza.

Scegliamo come soluzione la sequenza così generata e calcoliamo L_{max} a partire da tale ordinamento.

2.3 Ordinamento per tempo di processo: Prod

Dato un insieme di job e l'insieme dei loro processing time, disponiamo i job nella sequenza secondo l'ordine crescente dei loro processing time. Ossia

- Al primo posto nella sequenza c'è il job con processing time minore.
- All'ultimo posto nella sequenza c'è il job con processing time maggiore.

Scegliamo come soluzione la sequenza così generata e calcoliamo L_{max} a partire da tale ordinamento.
Come per l'algoritmo EDD (**2.1**), su Matlab si utilizza il vettore degli indici restituito dalla funzione $sort(...)$.

3 Euristiche iterative

Si propongono di seguito le tre euristiche da noi pensate per esplorare localmente lo spazio delle soluzioni, a partire da una soluzione data.
Ognuna di esse, in prima fase, si occupa di definire un vicinato, ossia un intorno di soluzioni centrato nella sequenza data. Questa prima fase è quella che vedete descritta, per ciascuna euristica, a seguire.
Tra le soluzioni offerte dal vicinato viene poi selezionata esclusivamente quella con lateness massima minore e, se migliorante rispetto alla soluzione di partenza, al termine dell'iterazione, ciascun algoritmo la sceglie come nuova soluzione corrente e riparte da essa con l'esplorazione del nuovo vicinato. Se ad una certa iterazione non venisse trovata, nell'intorno della soluzione corrente, alcuna soluzione migliorante, ciò signfica che la soluzione corrente è un minimo locale, e l'algoritmo termina.
Abbiamo quindi optato per metodi euristici di Ricerca Locale di tipo best-improvement.

3.1 Algoritmo latenessMax

Sia data una sequenza di partenza, eventualmente randomica. Un'iterazione di questo algoritmo definisce il vicinato nel modo seguente:

- Individua il job avente lateness massima nella sequenza, $JobL_{max}$ (evidenziato in fig. 1 tramite un asterisco).
- Se $JobL_{max}$ non è in prima posizione, sposta $JobL_{max}$ in ciascuna posizione precedente alla sua attuale, generando a ogni spostamento una diversa soluzione (vedi fig. 1). Altrimenti, restituisce un'unica sequenza coincidente con quella di partenza.

Nell'esecuzione del secondo punto tiene conto, al momento dello spostamento di $JobL_{max}$, di un eventuale "slittamento verso destra" di una posizione ad opera di alcuni job. Tale meccanismo di "slittamento" si capisce facilmente dall'esempio 3.4.

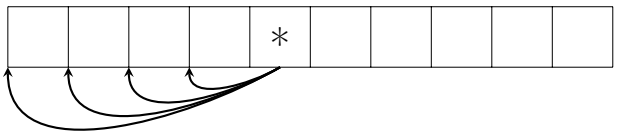

fig. 1

3.2 Algoritmo setupMax

Sia data una sequenza di partenza, eventualmente randomica. Una iterazione di questo algoritmo individua il job avente tempo di setup massimo nella sequenza, $JobSetUp_{max}$ (evidenziato in fig. 2 tramite un asterisco). Genera poi 3 soluzioni, ciascuna a partire dalla stessa sequenza iniziale:

- Se $JobSetUp_{max}$ è almeno in terza posizione, genera una prima soluzione effettuando lo scambio del job in due posizioni precedenti con quello subito precedente $JobSetUp_{max}$ (p_1).
- Se $JobSetUp_{max}$ è almeno in seconda posizione, genera una seconda soluzione effettuando lo scambio del job in posizione subito precedente $JobSetUp_{max}$ con $JobSetUp_{max}$ stesso (p_2).
- Se $JobSetUp_{max}$ non è in ultima posizione, genera una terza soluzione effettuando lo scambio di $JobSetUp_{max}$ con il job in posizione a lui subito successiva (p_3).

Si evita, in questo modo, di avere nella nuova sequenza il tempo di setup di $JobSetUp_{max}$, che risultava pesante temporalmente.

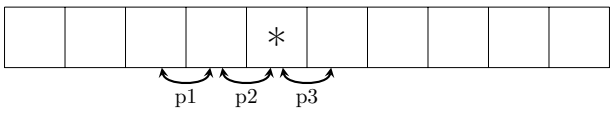

fig. 2

3.3 Algoritmo Rand*k*

Siano dati una sequenza di partenza, eventualmente randomica, e un numero naturale k. Un iterazione di questo algoritmo:

- Genera k nuove soluzioni, ognuna a partire dalla sequenza di partenza, scambiando di posizione 2 job scelti casualmente.

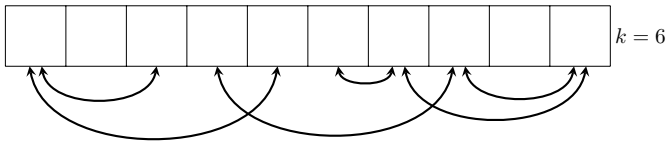

fig. 3

3.4 Esempio

Offriamo un esempio che meglio chiarisca il funzionamento dei tre algoritmi appena descritti.
Data la sequenza a seguire, dove supponiamo il job 1 sia quello con lateness massima, $JobL_{max}$, e il job 7 sia quello con tempo di setup massimo, $JobSetUp_{max}$, visualizziamo il vicinato generato rispettivamente dalle euristiche **3.1**, **3.2**, e **3.3**.

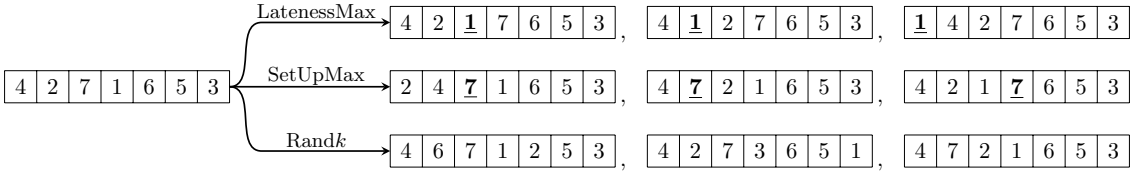

fig. 4

3.5 Approccio Multistart

Si possono generare molteplici sequenze iniziali e applicare ogni algoritmo iterativo sopra descritto parallelamente a tali sequenze. Si sceglie poi come soluzione corrente di ogni algoritmo la migliore tra le soluzioni parallelamente ottenute.

4 Euristiche Miste

Le euristiche costruttive (**2**) e iterative (**3**) funzionano meglio quando combinate (come si vedrà poi dalle analisi in **6.3.1**), ossia quando si applicano le euristiche iterative ad una soluzione generata tramite un'euristica costruttiva. Proponiamo dunque sette algoritmi frutto della combinazione tra le due tipologie di euristiche, cercando di migliorare progressivamente gli abbinamenti più efficaci.

4.1 Algoritmo EDD + latenessMax

A partire dalla soluzione generata tramite l'euristica costruttiva EDD, applica l'euristica iterativa latenessMax.

4.2 Algoritmo EDD + setupMax

A partire dalla soluzione generata tramite l'euristica costruttiva EDD, applica l'euristica iterativa setupMax.

4.3 Algoritmo EDD + Randk

A partire dalla soluzione generata tramite l'euristica costruttiva EDD, applica l'euristica iterativa Randk.

4.4 Algoritmo EDD + (setupMax + latenessMax)

A partire dalla soluzione generata tramite l'euristica costruttiva EDD:

1. Applica l'euristica iterativa setupMax (**3.2**)
2. Applica l'euristica iterativa latenessMax (**3.1**) alla soluzione restituita da 1.
3. Se la soluzione restituita da 2 è migliore di quella ottenuta da 1, pone come soluzione corrente quella restituita da 2 e torna allo step 1. Altrimenti, termina.

4.5 Algoritmo EDD + (latenessMax + setupMax)

A partire dalla soluzione generata tramite l'euristica costruttiva EDD:

1. Applica l'euristica iterativa latenessMax (**3.1**)
2. Applica l'euristica iterativa setupMax (**3.2**) alla soluzione restituita da 1.
3. Se la soluzione restituita da 2 è migliore di quella ottenuta da 1, pone come soluzione corrente quella restituita da 2 e torna allo step 1. Altrimenti, termina.

4.6 Algoritmo EDD + (setupMax + latenessMax + Randk)

A partire dalla soluzione generata tramite l'euristica costruttiva EDD:

1. Applica l'euristica iterativa setupMax (**3.2**)
2. Applica l'euristica iterativa latenessMax (**3.1**) alla soluzione restituita da 1.
3. Applica l'euristica iterativa Rand$k$ (**3.3**) alla soluzione restituita da 2.
4. Se la soluzione restituita da 3 è migliore di quella ottenuta da 1, pone come soluzione corrente quella restituita da 3 e torna allo step 1. Altrimenti, termina.

4.7 Algoritmo EDD + (setupMax + latenessMax) + Randk

A partire dalla soluzione generata tramite l'euristica costruttiva EDD:

1. Applica l'euristica iterativa setupMax (**3.2**)
2. Applica l'euristica iterativa latenessMax (**3.1**) alla soluzione restituita da 1.
3. Se la soluzione restituita da 2 è migliore di quella ottenuta da 1, pone come soluzione corrente quella restituita da 2 e torna allo step 1. Altrimenti, va a step 4.
4. Applica l'euristica iterativa Rand$k$ (**3.3**) alla soluzione restituita da 2.
5. Se la soluzione restituita da 4 è migliore di quella ottenuta da 2, pone come soluzione corrente quella restituita da 4 e torna allo step 1. Altrimenti, termina.

5 Modello MILP

Di seguito presentiamo il modello MILP da noi elaborato e implementato per il problema. Andando a cofrontare, per casi piccoli, i risultati dei nostri algoritmi con il risultato esatto al problema da esso evidenziato, il MILP sarà di grande utilità per certificare la qualità delle nostre euristiche.

5.1 Set

$N = \{1, 2, \ldots, n\}$ insieme dei job, con n numero totale di job
$\{0\}$ macchina vuota

5.2 Dati

p_i processing time del job $i \in N$
s_{ij} tempo di setup per passare dal job $i \in N$ al job $j \in N$, con $i \neq j$
s_{0i} tempo di setup iniziale del job $i \in N$
d_i data di consegna del job $i \in N$

5.3 Variabili decisionali

c_i tempo di completamento del job $i \in N$
u_i posizione nella sequenza del job $i \in N$
x_{ij} indica se il job $j \in N \cup \{0\}$ viene subito dopo il job $i \in N \cup \{0\}$, $\quad i \neq j$
δ_{ij} indica se, anche non consecutivamente, il job $j \in N$ segue il job $i \in N$, $\quad i \neq j$
L_{max} valore della lateness massima

5.4 Modello di ottimizzazione

Una possibile formulazione del modello è:

$$
\begin{aligned}
\min\ & L_{max} && && (1)\\
\text{s.t. } & L_{max} \geq c_i - d_i && i \in N && (2)\\
& \sum_{j \in N \cup \{0\}} x_{ij} = 1 && i \in N \cup \{0\} && (3)\\
& \sum_{i \in N \cup \{0\}} x_{ij} = 1 && j \in N \cup \{0\} && (4)\\
& x_{ii} = 0 && i \in N \cup \{0\} &&\\
& u_i - u_j + n x_{ij} \leq n - 1 && i, j \in N,\ i \neq j && (5)\\
& \delta_{ii} = 0 && i \in N &&\\
& \delta_{ij} = 1 - \delta_{ji} && i \in N,\ i \neq j && (6)\\
& u_j - u_i \leq \delta_{ij} M && i, j \in N,\ i \neq j && (7)\\
& c_i + p_j + \sum_{k \in N} x_{kj} s_{kj} \leq c_j + M(1 - \delta_{ij}) && i, j \in N && (8)\\
& p_j + x_{0j} s_{0j} \leq c_j && j \in N && (9)\\
& c_i \geq 0 && \forall i \in N &&\\
& u_i \in \mathbb{N} && \forall i \in N &&\\
& x_{ij} \in \{0, 1\} && \forall i, j \in N \cup \{0\} &&\\
& \delta_{ij} \in \{0, 1\} && \forall i, j \in N &&\\
& L_{max} \in \mathbb{R}. &&&&
\end{aligned}
$$

dove i vincoli stanno a significare che:
(2): la lateness massima è maggiore o uguale alla lateness di ogni job, che è per definizione la differenza tra il tempo di completamento e la data di consegna. Essendo un problema di minimizzazione è assicurata l'uguaglianza nel caso di job con lateness massima.
(3): ciascun job è immediatamente seguito, nella sequenza, da uno e un solo job. In particolare, il job in ultima posizione è immediatamente seguito dalla macchina vuota.
(4): ciascun job è immediatamente preceduto, nella sequenza, da uno e un solo job. In particolare, il job in prima posizione è immediatamente preceduto dalla macchina vuota.
(5): a ciascun job è assegnata la sua posizione nella sequenza. Inoltre, si evita la formazione di subtour.
(6),(7): δ_{ij} è pari a 1 sse il job $i \in N$ è precedente al job $j \in N$ ed è pari a 0 sse il job $i \in N$ è successivo al job $j \in N$.
(8): preso un job qualsiasi, per ogni job precedente (anche non immediatamente) vale la seguente disuguaglianza: il tempo di completamento del job in questione è maggiore o uguale alla somma del suo processing time, del suo tempo di setup rispetto al job subito precedente e del tempo di completamento del job precedente.
(9): il tempo di completamento del primo job della sequenza è maggiore o uguale al proprio processing time sommato al tempo di setup iniziale.

6 Confronto tra gli algoritmi

Confrontiamo di seguito gli algoritmi proposti sotto due aspetti principali: efficacia ed efficienza, dove, con efficienza, intendiamo quella temporale. Trascuriamo invece l'efficienza spaziale.

6.1 Euristiche costruttive e Modello MILP

6.1.1 Efficacia

Vengono effettuate più run dei diversi algoritmi costruttivi, variando la dimensione del problema. L'efficacia di ciascun algoritmo è valutata tramite il valore $L_{max,algoritmo}/L_{max,MILP}$.

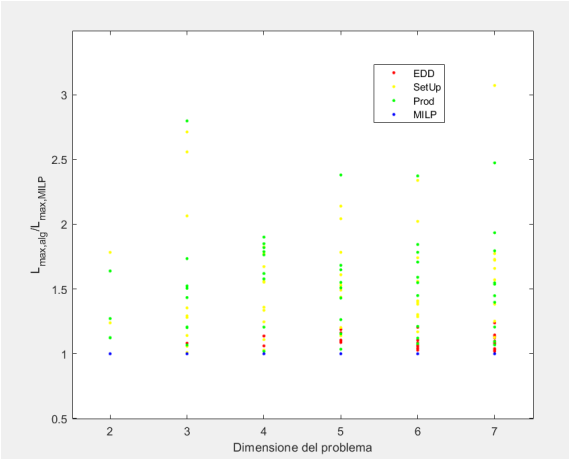

fig. 5

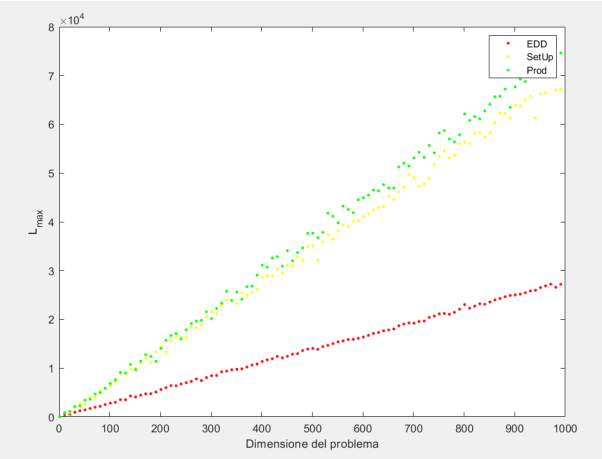

fig. 6

In fig. 5 si può notare come, tra gli algoritmi costruttivi, l'algoritmo più efficace sia EDD (**2.1**).
È poi possibile effettuare uno studio su dimensioni ancora maggiori, questa volta senza rapportarle alla $L_{max,MILP}$: è ciò che avviene in fig. 6 dove, a parità di dimensione, i dati di partenza sono gli stessi per ciascun algoritmo.
Si può nuovamente vedere come l'algoritmo EDD sia il più efficace tra quelli considerati.

6.1.2 Efficienza

Si confronta il tempo computazionale, rispetto alle dimensioni del problema, per le euristiche costruttive proposte e per il modello MILP. Sulle ordinate è riportato il tempo di esecuzione di ogni algoritmo, misurato in secondi.

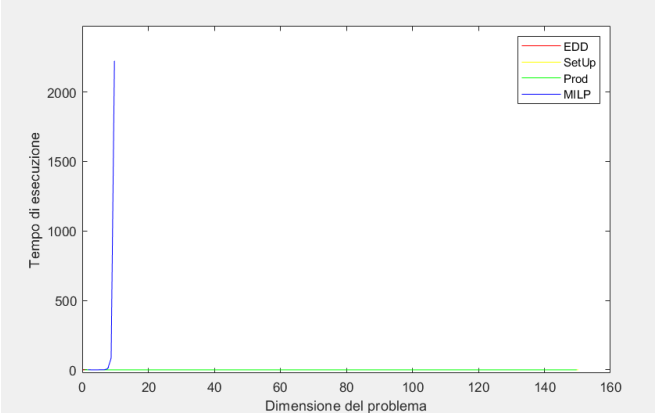

fig. 7

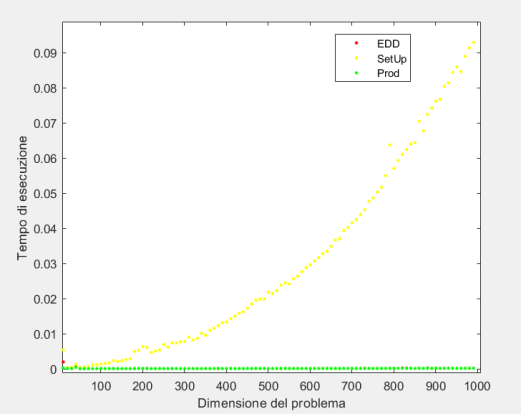

fig. 8

Si può notare in fig. 7 come la crescita estremamente rapida del tempo di esecuzione del modello MILP renda non apprezzabili i tempi di esecuzione delle euristiche costruttive. Si può ovviare al problema rimuovendo la voce MILP dalla rappresentazione grafica, come illustrato in fig. 8.
In quest'ultimo grafico il plotting di EDD (**2.1**) è sovrapposto, ad eccezione del primo punto, a quello di Prod (**2.3**). Da questi due plotting notiamo come il tempo di esecuzione degli algoritmi EDD e Prod, che devono la loro efficienza all'efficienza della funzione *sort* di Matlab, dipenda scarsamente dalle dimensioni del problema. Lo stesso non vale per l'algoritmo SetUp (**2.2**), invece, il cui tempo di esecuzione sembra avere un andamento quadratico.

6.1.3 Conclusioni

Il miglior compromesso efficacia-efficienza, tra le euristiche costruttive, è rappresentato dall'algoritmo EDD (**2.1**), che risulta essere il più buono per entrambi i parametri di valutazione.

6.2 Euristiche iterative

6.2.1 Efficacia

A partire da soluzioni casuali, vengono effettuate più esecuzioni dei diversi algoritmi, variando la dimensione del problema. L'efficacia di ciascun algoritmo è valutata tramite il valore $1-L_{max,algoritmo}/L_{max,iniziale}$, riportato sulle ordinate come percentuale.

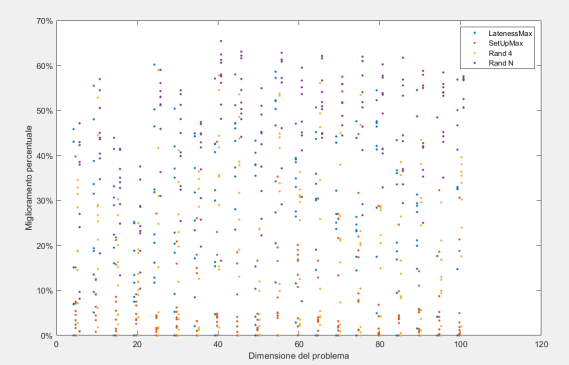

fig. 9

Dal grafico si possono trarre alcune considerazioni, riguardanti il miglioramento percentuale (quanto i pallini di un certo algoritmo sono in alto, nel grafico) e la costanza di tale miglioramento (quanto i pallini di un certo algoritmo sono sparsi tra di loro, nel grafico). Per esempio, l'algoritmo setupMax (**3.2**) porta a miglioramenti piccoli, perchè esplora un vicinato simile alla soluzione di partenza e di piccole dimensioni, che non scala con la dimensione del problema. L'algoritmo LatenessMax (**3.1**) porta a buoni miglioramenti, per quanto sia incostante, e dunque inaffidabile. Gli algoritmi Rand$k$ (**3.3**), invece, portano a miglioramenti mediamente maggiori, tuttavia sono decisamente incostanti. Inoltre si può notare nuovamente, confrontando i Rand$k$, come un vicinato che non scala con la dimensione del problema peggiori le sue prestazioni al crescere della stessa.

6.2.2 Efficienza

Vengono effettuate più esecuzioni a partire da soluzioni casuali dei diversi algoritmi, variando la dimensione del problema. Sulle ordinate è riportato il tempo di esecuzione di ogni algoritmo, misurato in secondi.

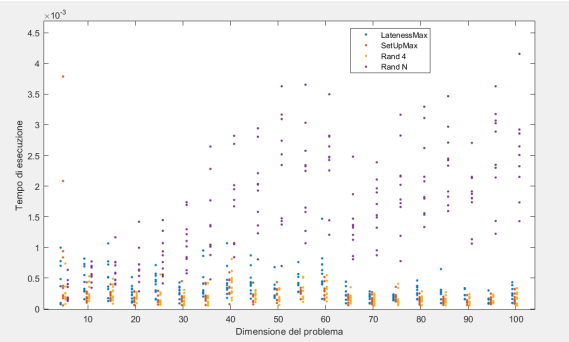

fig. 10

Si può osservare come il tempo di esecuzione di Rand$n$ (**3.3**) aumenti al crescere della dimensione del problema, visto che ad ogni iterazione l'algoritmo genera n vicini. Il tempo di esecuzione di LatenessMax (**3.1**), invece, per quanto il numero di vicini dipenda dalla dimensione del problema, sembra restare pressoché costante.

6.2.3 Algoritmo Rand$k$

Vediamo, al variare della dimensione del problema, come si comporta l'algoritmo Rand$k$ (**3.3**) con vicinati diversi. Per quanto riguarda l'efficacia, si ottiene quanto segue:

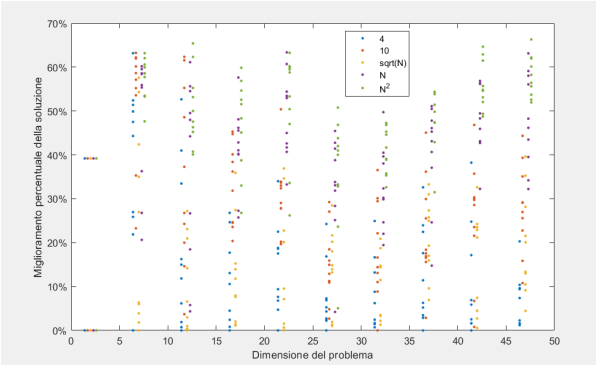

fig. 11

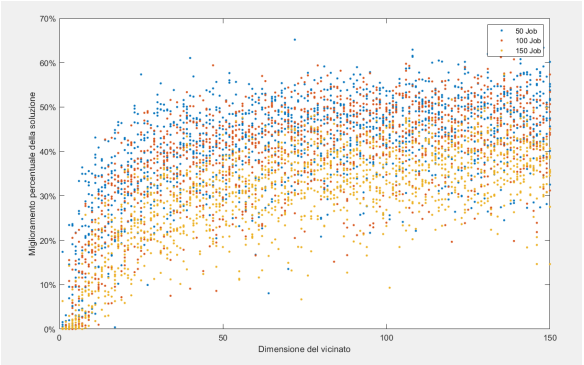

fig. 12

Ovviamente l'algoritmo Rand*4*, efficace per dimensioni del problema piccole, diventa inefficace al crescere delle stesse. Lo stesso vale per Rand*10*, che perde di efficacia da dimensioni maggiori rispetto a Rand*4*. Il vicinato di N^2 elementi si dimostra il migliore e il più affidabile. Il vicinato di N elementi, invece, testimonia una buona efficacia, ma non è affidabile. Un discorso analogo vale per l'algoritmo con $\sqrt{N}$, che è meno efficace, ma decisamente più veloce, come vedremo nei grafici a seguire.
È interessante notare che nella metà destra di fig. 12 il miglioramento, al crescere del vicinato, è praticamente costante. Pertanto, ci viene da ipotizzare che ricorrere a vicinati troppo grossi sia potenzialmente sconveniente, in quanto porti a un grosso consumo di risorse senza avere un particolare beneficio in termini di efficacia.
Per quanto riguarda i tempi di esecuzione, si trovano gli andamenti seguenti (nel grafico in fig. 14 il plotting di N^2 è stato rimosso per poter apprezzare gli altri).

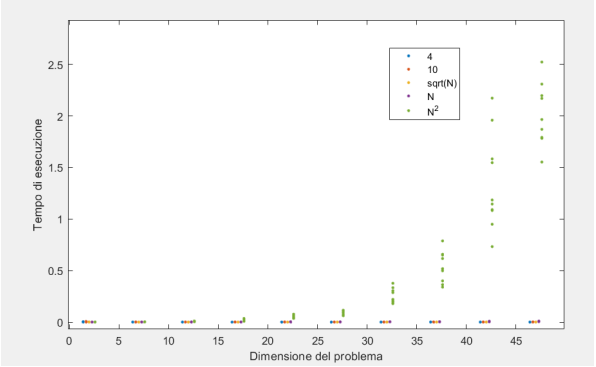

fig. 13

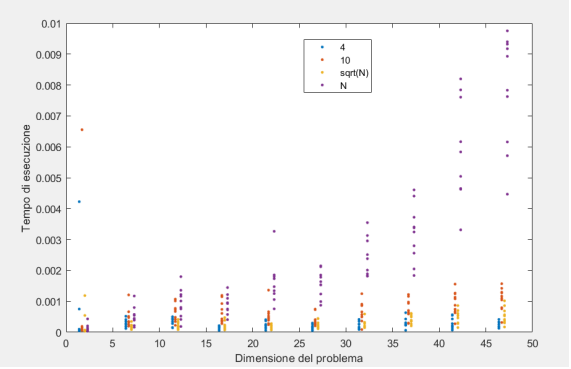

fig. 14

Come ci si poteva aspettare, il tempo di esecuzione dipende dalla dimensione del vicinato. Il vicinato di N^2 elementi abbiamo visto essere il migliore e il più affidabile ma possiamo notare che il costo computazionale è enorme.
In conclusione, per problemi di grosse dimensioni, potrebbe dare buoni risultati, come trade-off tra tempi e prestazioni, un algoritmo che crei un vicinato di dimensioni intermedie tra $\sqrt{N}$ ed N.

6.2.4 Multistart

Vediamo, al variare del numero di soluzioni di partenza, come si comporta una partenza multistart.

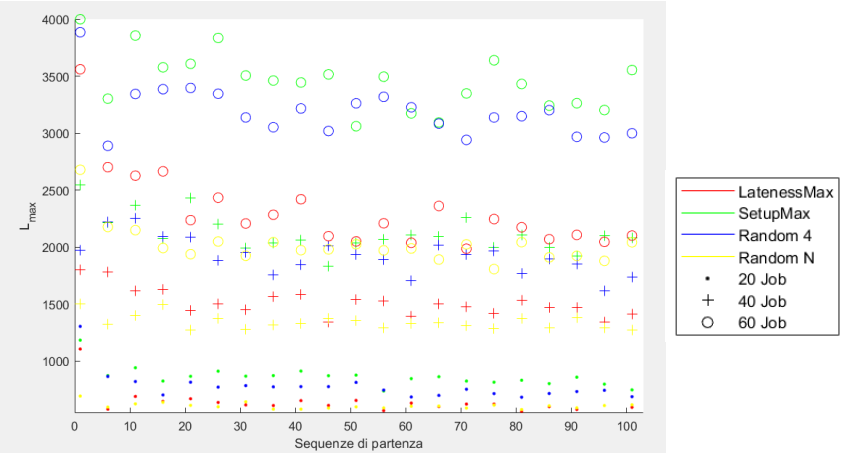

fig. 15

Si può osservare un miglioramento della soluzione al crescere del numero di sequenze di partenza. E' da considerare, però, che il tempo di esecuzione con l'approccio multistart aumenta linearmente all'aumentare del numero delle sequenze di partenza: ogni sequenza di partenza aggiunta aumenta il tempo di esecuzione di una quantità pari, circa, al tempo computazionale di una singola applicazione dell'euristica iterativa.

6.2.5 Conclusioni

Considerando il trade-off tra efficacia ed efficienza, probabilmente l'algoritmo migliore tra quelli proposti, partendo da soluzioni casuali, è LatenessMax (**3.1**). In alternativa, l'algoritmo Rand$k$ (**3.3**)con la creazione di un vicinato di dimensioni adeguate alla dimensione del problema, in modo da trovare un equilibrio tra percentuale di miglioramento e tempo di esecuzione.

6.3 Euristiche Miste

6.3.1 Iterative vs EDD+Iterative

Vediamo, al variare della dimensione del problema, come si comportano le euristiche iterative a partire da una sequenza casuale rispetto alle stesse partendo dalla sequenza generata da EDD (**2.1**).

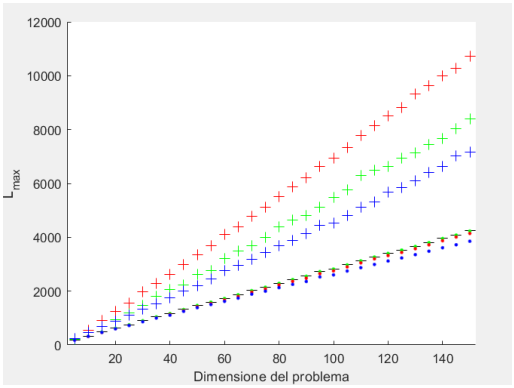

fig. 16

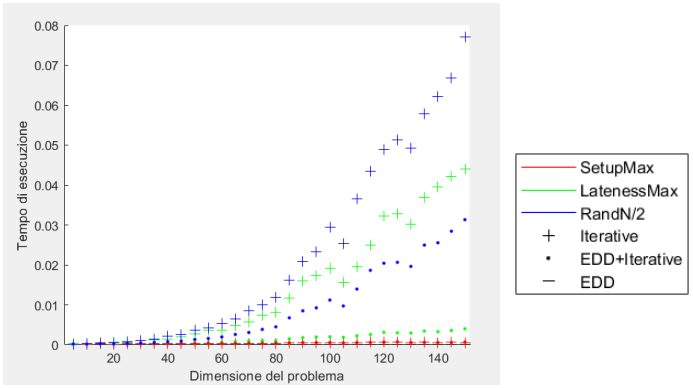

fig. 17

Si osserva immediatamente come le euristiche iterative applicate a sequenze casuali siano nettamente peggiori già soltanto di EDD. Si può inoltre notare come partendo dalla sequenza EDD si ottengano risultati migliori in tempi minori, a dimostrazione della schiacciante superiorità delle euristiche miste rispetto alle iterative prese singolarmente. I tempi minori sono verosimilmente dovuti al numero inferiore di iterazioni che ogni algoritmo effettua prima di terminare, dato che l'esecuzione comincia da una sequenza già di partenza migliore rispetto ad una casuale.
L'euristica iterativa SetupMax (**3.2**), che partendo da una sequenza casuale risultava sconveniente, paragonata a LatenessMax (**3.1**), partendo invece da EDD (**4.2**)risulta più efficace di LatenessMax (**4.1**), come vedremo meglio in **6.3.2**. La motivazione di questo comportamento può essere attribuita al criterio di esplorazione del vicinato dei due algoritmi: dato che l'euristica costruttiva EDD tiene in considerazione

solo le date di consegna, SetupMax consente di cambiare "direzione di esplorazione" e cerca di migliorare la soluzione da un altro punto di vista. Al contrario, LatenessMax è nuovamente focalizzata sulle date di consegna e cerca di migliorare ulteriormente sequenze già molto buone sotto questo punto di vista. Osservando anche il grafico dei tempi di esecuzione, concludiamo dunque che, probabilmente, la scelta migliore come trade-off efficacia-efficienza ricada su EDD+SetupMax (**4.2**).

6.3.2 Miste vs Miste

Qui di seguito confrontiamo efficacia ed efficienza delle euristiche miste, tramite rappresentazioni grafiche ed esempi numerici.

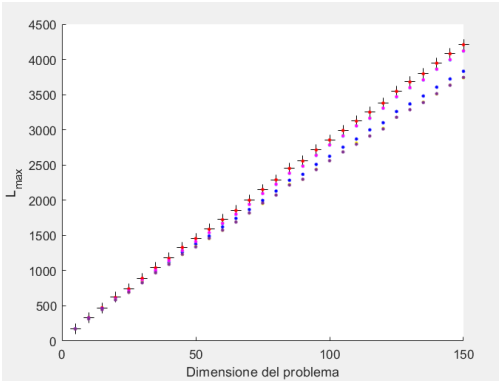

fig. 18

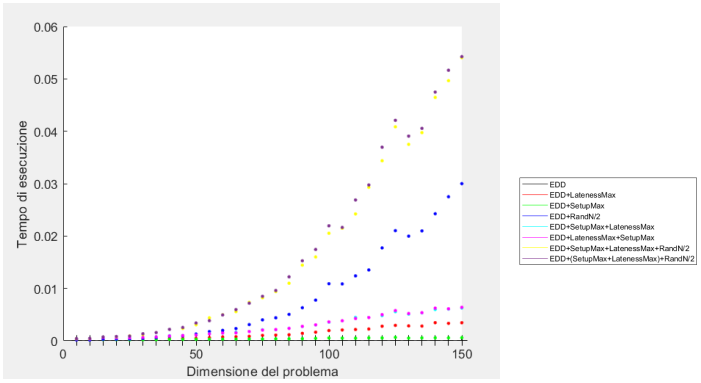

fig. 19

Attenzione: il rosa in fig. 18 è sovrapposto al verde e all'azzurro, e il viola è sovrapposto al giallo.

Come ci si poteva aspettare, maggiore è il numero di euristiche iterative utilizzate, maggiore è il tempo di esecuzione della mista, ma anche migliore è il risultato ottenuto. La combinazione EDD+(setupMax+latenessMax)+Rand$k$ (**4.7**) risulta la migliore in termini di efficacia, ma la peggiore in termini di efficienza; al contrario, l'euristica EDD+latenessMax (**4.1**) risulta la peggiore in efficacia, ma tra le migliori in efficienza. Dunque, la scelta dell'algoritmo adatto dipende molto dalla dimensione del problema e dalle esigenze del fruitore. Con un numero elevato di job, l'algoritmo migliore come rapporto efficacia-efficienza probabilmente risulta essere EDD+setupMax (**4.2**), imbattibile dal punto di vista del tempo di esecuzione, ma non peggiore in efficacia di EDD+latenessMax, EDD+(setupMax+latenessMax) (**4.4**) e EDD+(latenessMax+setupMax) (**4.5**).

Di seguito una tabella contenente alcuni parametri ottenuti dall'esecuzione di EDD (**2.1**) ed ogni euristica mista. Nelle colonne troviamo: la lateness massima media, il miglioramento percentuale medio rispetto ad EDD e il tempo medio di esecuzione dell'algoritmo, calcolati a partire da 10000 esperimenti con numero di job $n = 100$ e numero di scambi dell'algoritmo Rand $k = 50$.

Euristiche	L_{max}	Miglioramento (%)	Tempo (s)
EDD	2841	-	7.23562e-05
EDD + latenessMax	2838	0.12	0.0039901
EDD + setupMax	2768	2.59	0.0009073
EDD + Rand	2618	7.84	0.0213657
EDD + (setupMax + latenessMax)	2764	2.73	0.0073578
EDD + (latenessMax + setupMax)	2766	2.65	0.0075767
EDD + (setupMax + latenessMax + Rand)	2548	10.31	0.0442596
EDD + (setupMax + latenessMax) + Rand	2549	10.27	0.0446801

7 Conclusioni finali e considerazioni

Le euristiche analizzate in questo report sono solo una piccola selezione di quelle possibili. Ne abbiamo dunque scelte alcune che permettessero di ottenere risultati buoni, ma che avessero anche tempi di esecuzione sostenibili. Abbiamo quindi dato la priorità a trarre dallo studio di queste euristiche delle osservazioni di carattere generale, piuttosto che a creare l'algoritmo più efficiente possibile.
Per quanto riguarda il codice Matlab, anche se poteva essere compattato ulteriormente, abbiamo deciso, in alcuni punti, di tenere la struttura ideata originariamente, anche per chiarezza alla lettura. In quanto al modello MILP, invece, ci siamo resi conto che la versione per cui abbiamo optato inizialmente poteva essere riformulata in maniera più compatta, diminuendo il numero di variabili decisionali in gioco. Tuttavia, ai fini del problema, abbiamo deciso di mantenere la versione originale, che è quella che vedete riportata.