RELAZIONE FINALE
Team 3 - LeMur

1. Descrizione della Challenge e Obiettivi

La programmazione lavorativa dei dipendenti, come la gestione dei turni lavorativi o del carico di lavoro assegnato, è da sempre stata un aspetto cruciale all'interno dell'organizzazione di un'azienda.
Questo risulta nella creazione di un calendario che permetta una buona efficienza lavorativa e rispetti le varie scadenze imposte dai clienti (anche a fronte di imprevisti). Il tutto senza compromettere un ambiente sano in cui i lavoratori abbiano un carico adeguato di lavoro, riducendo la possibilità di errori nella produzione dovuti alla stanchezza accumulata.

Il compito è però risolvibile in maniera ottimale grazie a programmi (i cosiddetti "scheduler") che hanno il compito di trovare, attraverso specifici algoritmi di ricerca, una programmazione lavorativa secondo diversi parametri, come efficienza, velocità produzione o distribuzione del carico lavorativo, basata sulle specifiche richieste di ogni azienda.

LeMur affida al Team 3 dell'Industrial AI Challenge il compito di creare questo tipo di software, in grado di suggerire una possibile programmazione del reparto spiralatura, il quale presenta svariate dinamiche e complessità, come interventi manuali e operazioni asincrone, che vanno quindi definite e modellate appropriatamente.

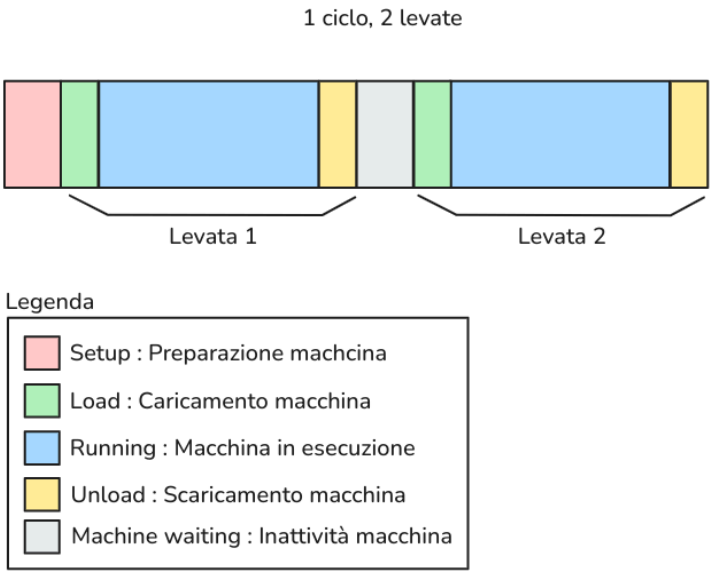
2. Definizione del problema

Per soddisfare le aspettative dell'azienda, le prime settimane della challenge sono state dedicate a comprendere in modo esaustivo come LeMur gestisse i propri processi interni e quali fossero i loro obiettivi. Risulta vitale dunque, definire il vocabolario e gli obiettivi rilevanti per la riuscita della sfida. I concetti fondamentali che abbiamo identificato sono i seguenti:

- **Articolo**: la materia finale del processo produttivo, caratterizzata da specifiche tecniche, le quali forniscono i dettagli inerenti alla sua produzione. In particolare siamo interessati alle implicazioni che differenti prodotti hanno in relazione ai **macchinari** (mezzi tramite i quali viene composta la materia "grezza"). Dettagli importanti sono i seguenti:
    a. Un articolo può essere processato solo in un sottoinsieme di macchinari.
    b. Preso il sottoinsieme di macchine compatibili con un qualche prodotto, la produzione di Kg/h differisce fra questi, principalmente per via delle diverse capacità tra i macchinari, identificata dal numero di **fusi**.

- **Ciclo**: Il processo eseguito da un macchinario per produrre una data quantità di prodotto, è riferito come unità produttiva. Viene suddiviso ulteriormente in **levate** ed è strutturato nel modo seguente:
    a. **Setup**: Preparazione macchina, comporta il caricamento dell' **elastomero** e l'impostazione della **velocità** (definita in relazione all'articolo da produrre, con un certo margine del ±5%)
    b. **Load**: Caricamento macchina.
    c. **Running**: Macchina in esecuzione, ha una durata minima definita in relazione al prodotto e alla velocità impostata.
    d. **Unload**: Scaricamento macchina.

e. **Machine waiting**: inattività macchina. All'interno di un ciclo, è presente principalmente nella fase running.
N.B: all'interno del documento si fa riferimento alle operazioni di: setup, load e unload anche con il termine "interventi manuali". Questo poiché appunto sono operazioni che richiedono supporto degli operatori per essere effettuate



- **Operatori**: lavoratori LeMur con il compito di eseguire e supervisionare gli interventi manuali. Lavorano suddivisi in gruppi di dimensione definita e omogenea (nella nostra formulazione).
- **Prodotto**: Nella nostra formulazione rappresenta l'insieme di tutti i cicli relativi ad un qualche articolo richiesto da un cliente. Ad esempio: 2 clienti che richiedono lo stesso articolo risulteranno in due prodotti differenti, ognuno con i propri cicli associati.

Riguardante gli obiettivi richiesti da LeMur, questi possono essere riassunti nei punti seguenti:

1. La programmazione dell'attività delle macchine deve ricercare la **massima efficacia possibile**, cercando quindi, data una previsione degli articoli da produrre, di aver meno tempi morti possibili e che termini con il **maggior margine temporale**, permettendo quindi di essere robusti ad eventuali imprevisti.
2. Le operazioni che richiedono un intervento umano rappresentano la maggior criticità nella pianificazione. È importante che queste siano distribuite uniformemente, **evitando sovraccarichi giornalieri**. Abbiamo dunque convenuto che ci fosse la necessità di limitare il numero di interventi umani nello stesso istante temporale, fissandolo come input del sistema.
3. **La capacità di creare un programma flessibile** che possa pianificare la produzione attorno ad esigenze particolari quali, ad esempio, manutenzioni ordinarie e straordinarie, o danni alla catena di produzione.

4. I punti precedenti sono dunque rappresentati dalla creazione di una **programmazione completa** come obiettivo ultimo. Si consideri uno stato attuale comprensivo di: un insieme di ordini da programmare ed un insieme di ordini attualmente esecuzione su dei macchinari. Venga definito un orizzonte temporale all'interno del quale si ricerchi una soluzione (se esistente) che preveda il raggiungimento del quantitativo di Kg richiesto dal cliente, senza violare una serie di vincoli imposti. Vantaggio importante rispetto all'attuale pianificazione di LeMur è proprio la componente temporale, la quale permette lungimiranza nel lungo periodo, e non solo in quello breve adottato attualmente (settimanale/giornaliero).

3. Metodi e Strumenti

In letteratura, la programmazione richiesta da LeMur è meglio nota come **Job Shop Problem (JSP)** [3] [4]. Come altri problemi del suo genere, fornisce una formulazione matematica di base, sulla quale è possibile costruire una serie di ulteriori vincoli matematici (constraints), permettendo di personalizzarlo alle caratteristiche del reparto spiralatura in questo caso.

I modi più comuni di approcciare JSP ricadono sotto 3 categorie principalmente:

(1) Constraint Programming (integer, mixed-integer, …)
(2) Bio-inspired (Genetic algorithms, ant colony optimization, …)
(3) Reinforcement Learning

I metodi della categoria (3) sono in assoluto i più orientati al dato, utilizzando tecniche moderne di Machine Learning e Deep Learning. Per quanto allettanti, il problema intrinseco sta proprio nel dato stesso. Questi approcci richiedono almeno svariati milioni di dati e non sono quindi adatti al nostro contesto, avendo a disposizione una base di dati non così numerosa. Inoltre, un importante aspetto individuato per il nostro programma è quello di garantire la correttezza delle soluzioni proposte, vincolo non sempre assicurato da sistemi tipo (3).

Adottare esclusivamente una strategia di tipo (2) è decisamente realistico seguendo la letteratura. La complessità implementativa tuttavia cresce notevolmente al crescere delle specifiche del problema.

I metodi di tipo (1) invece sono in un certo senso "esatti". Il principio è relativamente semplice, poiché ci si "limita" a definire i vincoli matematici e si "delega" la ricerca della soluzione ad un cosiddetto **solver**. Questo internamente andrà alla ricerca di una soluzione che rispetti i vincoli imposti. Talvolta (1) è utilizzato in congiunzione con (2) andando a costruire sistemi di ricerca ancora più evoluti, complessi e flessibili. La nostra scelta di design ricade proprio in quest'ultimo tipo.

A seguito di una revisione della letteratura inerente, abbiamo convenuto sull'utilizzo di *Google OR-Tools* [1], libreria che permette la definizione e risoluzione di problemi di Constraint Programming (CP) e non solo.
Malgrado l'ottimo bilanciamento fra semplicità implementativa ed efficacia nella risoluzione offerto dalla libreria, il problema di LeMur rimane "NP-completo". Per spiegare cosa sia un problema NP-completo, introduciamo brevemente il termine "complessità" e soprattutto la

classe di complessità denominata “NP”. In informatica, per complessità si intende la quantità di risorse (tempo, memoria, traffico sulla rete, ecc.) spese dall’algoritmo per raggiungere l’obiettivo prefissato dallo sviluppatore. La categoria di complessità “NP”, racchiude tutti quei problemi, per i quali una macchina deterministica di Turing (un computer), richiede un tempo non deterministico e polinomiale per raggiungere la soluzione. Traducendo: al crescere dell’input del problema le risorse usate dal programma crescono in modo esorbitante, e non esiste, a oggi, un metodo che possa risolvere la cosa in maniera universale.

Preso atto di ciò, la nostra scelta di adottare una tecnica ibrida di (1) e (2) è volta proprio a limitare l’impatto computazionale del problema, permettendo di raggiungere migliori soluzioni in minor tempo. Nello specifico la componente (2) denominata da noi “rifinimento genetico” si basa appunto su un algoritmo di ricerca genetica, sviluppato con la libreria *Inspyred* [2].

4 Formulazione del problema

Inizieremo ora dando una breve introduzione teorica dei vari punti su cui abbiamo lavorato durante questi 2 mesi.

4.1 Constraint Programming

In informatica e più specificamente nel contesto di "logical reasoning” vengono definiti "constraints" tutti quei vincoli, che il programma deve rispettare o almeno cercare di soddisfare per discriminare le soluzioni trovate come candidati possibili. Questi “constraints” si dividono in due tipi.

4.1.1 Hard constraints

Delineano tutti quei vincoli la cui soddisfazione è obbligatoria. Soluzioni che non rispettino anche solo una di queste condizioni sono da considerarsi non valide.
Un esempio nel nostro caso può essere la richiesta da parte di LeMur di non superare il massimo di N interventi manuali in contemporanea. Il nostro software ignorerebbe qualsiasi possibile risultato che ne preveda un numero superiore.

4.1.2 Soft constraints

Rappresentano tutte quelle preferenze che una soluzione deve ricercare per essere presentata come miglior soluzione in assoluto, ciò che, a livello teorico in ottimizzazione, viene delineato come “minimo globale” o ottimo.
A differenza dei *hard constraints,* tali non impongono una condizione *necessaria* ma una *preferenza*. Una programmazione che soddisfi parzialmente questi punti può comunque rappresentare un ottimo.
Come nel precedente tipo, anche qui mostreremo un esempio pratico legato alle richieste fatte da LeMur. Una levata, eseguita su una qualche macchina, per un qualche articolo, ad una determinata velocità, ha una durata minima prestabilita, ma una durata effettiva variabile. Tipicamente soluzioni che hanno un arco temporale fra le operazioni di carico e scarico proprio uguale a quel costo minimo sono preferibili, tuttavia, è necessario che ci sia

la possibilità di avere dei momenti di attesa, banalmente per permettere al personale di essere disponibile, o perché la produzione termina durante la notte.

4.2 Constraints e Variabili specifiche di LeMur

I vincoli in un problema come il JSP vengono definiti attraverso una serie di variabili matematiche d'appoggio, messe in relazione fra loro tramite operatori logici. Per questa ragione, introduciamo brevemente il significato di alcune variabili adottate da noi che differiscono lievemente dalla programmazione attuale di LeMur, ma fondamentali ai fini della creazione del modello:

- **Ciclo attivo / Inattivo**: indica se un ciclo viene eseguito o meno.

Questa suddivisione che può sembrare banale è assolutamente necessaria per via di una semplice questione: la capacità produttiva (Kg/h) fra le macchine compatibili con un prodotto NON è uniforme. Ciò comporta che per soddisfare il quantitativo richiesto da un cliente si potrebbero, ad esempio, eseguire 2 cicli su una macchina, o soltanto 1 in un'altra (compatibile) avente il doppio della capacità produttiva. Nella prospettiva di JSP questa è una complicazione notevole, poiché nella formulazione base [5] [6] [7] il numero di Jobs (cicli nel nostro caso) è una costante in input al problema; assunzione che invece non possiamo considerare nel nostro contesto, diventando un parametro su cui è richiesta una ricerca esplicita.

Similmente, un ciclo potrebbe non richiedere l'esecuzione completa di tutte le levate nominali, in quanto questo porterebbe ad una sovrapproduzione eccessiva rispetto alla richiesta del cliente. Per questo motivo abbiamo il bisogno di inserire altre 2 variabili su cui dover eseguire una ricerca esplicita rispetto alle formulazioni standard di JSP, quali:

- **Levata attiva / Inattiva**: indica se una levata viene eseguita o meno.
- **Ciclo completo / parziale**: indica se tutte le levate del ciclo sono attive (ciclo completo), o se vi sono presenti levate sia attive che inattive (ciclo parziale). Un ciclo che presenta solo levate inattive è naturalmente un ciclo inattivo.

Noti questi concetti da noi introdotti, passiamo ai vincoli da noi adottati:

1. **Operatori**: Non più di N interventi manuali in contemporanea;
2. **Operatori**: Ogni intervento manuale relativo al setup di cicli attivi deve avere esattamente un gruppo di operatori assegnato;
3. **Operatori**: Ogni intervento manuale relativo al setup di cicli inattivi non deve avere alcun gruppo di operatori assegnato;
4. **Operatori**: Ogni intervento manuale di una levata attiva deve avere esattamente un gruppo di operatori assegnato;
5. **Operatori**: Ogni intervento manuale di una levata inattiva non deve avere alcun gruppo di operatori assegnato;
6. **Operatori**: Gli interventi manuali hanno una durata minima, dipendente da: numero di operatori nel gruppo assegnato, numero di fusi della macchina.
7. **Operatori**: Gli interventi manuali possono essere spezzati fra un giorno lavorativo ed il successivo (anche attraverso il weekend / altre festività). Tale meccanismo è denominato nel codice come "gap".

8. **Operatori**: In ogni istante temporale, un gruppo di operatori non può essere associato a più interventi manuali contemporaneamente.
9. **Macchine**: Un ciclo attivo deve essere assegnato ad una e una sola macchina;
10. **Macchine**: Un ciclo inattivo non deve avere macchine assegnate;
11. **Macchine**: Ogni macchina, in ogni istante temporale, deve essere assegnata al più ad un ciclo;
12. **Produzione**: Al massimo un solo ciclo parziale per prodotto;
13. **Produzione**: I cicli completi devono essere anche attivi;
14. **Produzione**: I cicli parziali devono essere attivi ma non completi;
15. **Produzione**: Se il ciclo è completo allora il numero di “levate” completate è il massimo;
16. **Produzione**: Se il ciclo non è attivo allora il numero di “levate” completate deve essere 0;
17. **Produzione**: Ogni ciclo di un qualche prodotto deve avvenire dopo la data di inizio inserita;
18. **Produzione**: Ogni ciclo di un qualche prodotto deve terminare entro la data di consegna;
19. **Produzione**: Ogni prodotto deve soddisfare almeno il quantitativo di Kg richiesto, con un piccolo margine sulla sovrapproduzione;

Tutti i vincoli elencati in precedenza ricadono nella categoria di hard constraints in quanto, come già spiegato in precedenza, questi devono essere soddisfatti al fine di rendere la soluzione valida.
Nella formulazione finale del programma, non sono presenti soft-constraints. Ma siamo comunque riusciti a soddisfare tutte le richieste di LeMur per un prodotto funzionale.
Le possibili estensioni che richiederebbero l’utilizzo di soft-constraints con minime modifiche alla formulazione attuale del programma, in linea con gli obiettivi di lemur, sono riguardanti il dispendio energetico legato alla velocità di levata delle macchine.

4.3 Formulazione

Il modello scelto **non prevede alcuna forma di apprendimento dal dato, al contrario si focalizza sulla ricerca**. Dunque, il nostro approccio è categorizzabile come Machine Learning? Decisamente no: nel ML l’enfasi è prettamente sull’apprendere conoscenza dal dato stesso. Il nostro approccio è Artificial Intelligence? Assolutamente, poiché comprende una moltitudine di aspetti che vi ricadono; in particolare: ricerca, logica del primo ordine, nonché una nota genetica.
Per ribadire nuovamente il concetto: il nostro sistema non punta ad utilizzare esplicitamente i dati provenienti dallo storico per imparare il metodo di programmazione attualmente utilizzato da LeMur, al contrario, basandosi sullo stato attuale del sistema (ordini in ingresso, ordini già in esecuzione su macchinari, data odierna…), compie scelte atte a trovare una programmazione il più ottimale possibile.

Preso il modello matematico, rappresentato dall'insieme delle variabili e vincoli, è importante definire il nostro approccio alla ricerca di una soluzione, il quale prevede un ibrido di CP-SAT e Genetic Algorithms [8], suddiviso in 3 fasi di ottimizzazione:

1. **Makespan minimization**: CP-SAT ricerca una soluzione che minimizzi il Makespan (che rappresenta il tempo impiegato dalle macchine per completare i rispettivi cicli di produzione) globale fra le macchine. Questo è descritto dal massimo istante temporale fra tutti i termine ciclo.

$$max\ (FineCiclo)$$

2. **Compactness maximization**: considerato l'output di (1) CP-SAT fissa gli assegnamenti dei cicli alle macchine, andando a massimizzare la compattezza (che è semplicemente definito come l'inverso della durata dei cicli) di tali assegnamenti, in modo da avere soluzioni che presentano periodi di inattività minori in ogni macchinario. Ciò è rappresentato dalla somma di tutti gli istanti in cui terminano tutti i cicli.

$$\sum_i FineCiclo_i$$

3. **Genetic Refinement**: considerato l'output di (2) un algoritmo di ricerca genetica applica operazioni di mutazione volte a spostare / scambiare i cicli in altre macchine compatibili, diverse da quelle fissate in (2), in modo tale da ottimizzare la distribuzione dei cicli fra molteplici macchine. La funzione obiettivo di questa fase è identica a quella adottata in (2) con, in aggiunta, dei fattori di penalità per scoraggiare soluzioni che producono un makespan globale maggiore di quello trovato in (1).

$$\sum_i (FineCiclo_i) - penalità$$

Suddividere la ricerca in diverse fasi ha svariati vantaggi poiché riduce le risorse richieste notevolmente senza impattare in maniera eccessiva sulla qualità delle soluzioni.
L'effetto di ognuna di queste componenti è esplorato più nel dettaglio nella sezione 7 comprendente risultati quantitativi e, in parte, qualitativi.

5. Dati LeMur

Essendo l'approccio scelto da noi indipendente dall'apprendimento sul dato, i dati forniti da Lemur (in particolare lo storico) hanno un ruolo prettamente di supporto, volto a permetterci di comprendere quali siano i dati a nostra disposizione, su cui basare le variabili del software stesso.
La base di dati risulta poco curata per il nostro obiettivo, in parte incompleta, con svariate ridondanze, ambiguità e generalmente scarsa (pochi record). Al fine di estrapolare informazioni effettivamente utili molto del lavoro iniziale è stato dedicato alla pulizia di questi dati.

Lo storico contiene in particolare:

- Nome articolo;
- Macchina assegnata alla produzione del suddetto articolo;
- Cicli richiesti per la completamento del prodotto;
- Quantità di prodotto richiesto;
- Data in cui l'ordine è stato accettato;
- Data di consegna;
- Composizione del prodotto.

Pur avendo un'importanza marginale, questi dati ci hano permesso di estrarre in maniera automatica interrelazioni importanti, quali:

a. Compatibilità Articolo - Macchina
b. Statistiche Generali di richiesta e produzione dei prodotti

Al fine di testare in fase preliminare la nostra soluzione, le informazioni estratte sono state utilizzate al fine di creare un generatore di dataset sintetici. Questo ci ha permesso di creare ordini fittizi conformi a degli scenari plausibili per LeMur indispensabili per la validazione del sistema.

5.1 Inputs del sistema

Parliamo ora dei dati che il sistema richiede in input per produrre la tabella lavorativa:

- *new_orders.csv*: un file contenente una lista degli articoli che devono essere prodotti. Nello specifico il documento è composto dai seguenti dati:
  - cliente;
  - codice articolo;
  - quantità richiesta;
  - data inserimento, ovvero la data in cui l'ordine è stato accettato;
  - data consegna;
- *running_orders.csv*: presenta al suo interno informazioni relative agli ordini già avviati, come:
  - cliente;
  - coda articolo;
  - quantità;
  - fine operazione attuale, ovvero la fine del ciclo corrente;
  - data consegna;
  - levate rimanenti al termine del ciclo;

    - tipo di operazione attuale, ad esempio: levata, carico, scarico, ecc.;
    - operatore;
- *lista_articoli.csv*: fornisce specifiche sulla produzione di ogni articolo, nel dettaglio quest sono:
    - coda articolo;
    - kg/ora;
    - numero cicli, che indica il numero di cicli da compiere per raggiungere la quantità richiesta dal cliente;
    - ore levata, che rappresenta l'intervallo di tempo tra una levata e la successiva;
- *articoli_macchine.json*: definisce la compatibilità tra articoli e macchine (in termini tecnici, definisce in quali macchine i vari articoli sono stati posizionati in passato, secondo lo storico fornitoci);
- *macchine_info.json*: contiene dettagli riguardanti le specifiche di ogni macchina, come il numero di fusi.

6. Sviluppo settimanale

Di seguito un riassunto settimanale, al fine di evidenziare gli step seguiti e le difficoltà riscontrate. Si noti che ad ogni settimana è stato svolto un meeting congiunto con i mentori del progetto e i rappresentanti di LeMur, al fine di presentare e validare risultati intermedi, definire i prossimi passi da seguire e discutere di eventuali problemi o criticità.

(sett. 1) Prima riunione di kick-off con sessione Q&A. Analisi del contesto aziendale di LeMur con definizione delle prime limitazioni di base del problema. Discussione sulla struttura del progetto e le possibili direzioni con relativa identificazione della categoria di problemi più adatta al contesto

(sett. 2) Il team si divide in tre sottogruppi:

- Gruppo 1 e 2 per formulazione iniziale indipendente del problema da una prospettiva matematica/logica.
- Gruppo 3 per l'estrazione di metriche e componenti utili dai dati aziendali per supportare il lavoro degli altri gruppi.

(sett. 3) Lavoro sulle formulazioni matematiche e sviluppo di una prima versione di estrazione di informazioni dallo storico. Visita aziendale per confermare le conoscenze raccolte e la validità del lavoro in corso

(sett. 4) Redazione del documento di assunzioni e requisiti, al fine di rispettare le specifiche del problema aziendale.

(sett. 5) La formulazione matematica presenta un difetto importante che rende necessarie importanti modifiche ad essa, poiché parte delle ipotesi iniziali si sono rivelate errate. Pulizia della base di dati aziendali ed estrazione statistiche.

(sett. 6) Intenso lavoro sulla nuova formulazione matematica, tenendo conto delle nuove ipotesi e dei vincoli.

(sett. 7) Conferma validità della nuova formulazione matematica con i rappresentanti LeMur e definizione della nuova strategia operativa:

- esplorazione di un approccio bio-ispirato (gruppo 1).
- ottimizzazione della versione corrente con test su dati più realistici. Aggiunta di ultime funzionalità e definizione di strategie/euristiche per accelerare la ricerca delle soluzioni (gruppo2).
- sviluppo di un'applicazione di benchmark per mostrare i miglioramenti rispetto alla situazione precedente di LeMur (gruppo 3).

(sett. 8) Continuazione del lavoro indipendente dei gruppi seguendo la strategia operativa.

(sett. 9) Implementazione di alcune ultime funzionalità mancanti. Finalizzazione del lavoro sull'approccio genetico per perfezionare le soluzioni dello scheduler. Sviluppo dell'applicazione di benchmark.

(sett. 10) Continuazione del lavoro indipendente dei gruppi seguendo la strategia operativa.

(sett. 11) Sviluppo di un applicativo web per l'utilizzo da parte di LeMur. Stesura delle linee guida per l'inserimento dei dati in input e per l'esecuzione dello scheduler.

Per maggiori dettagli sulle attività svolte consultare la Miro Board.

7. Risultati

In questa sezione presentiamo i risultati ottenuti con attenzione prima al lato prestazionale, poi a quello di confronto infine alla validazione dei risultati ottenuti.

7.1 Performance

Per valutare le performance del nostro solver in modo completo, adottiamo misure quantitative sul funzionamento e le soluzioni dello scheduler, come il tempo impiegato per trovare una soluzione o la distribuzione del carico di lavoro lungo un arco temporale.
Esamineremo in seguito più nello specifico diverse statistiche rilevanti in questo contesto.

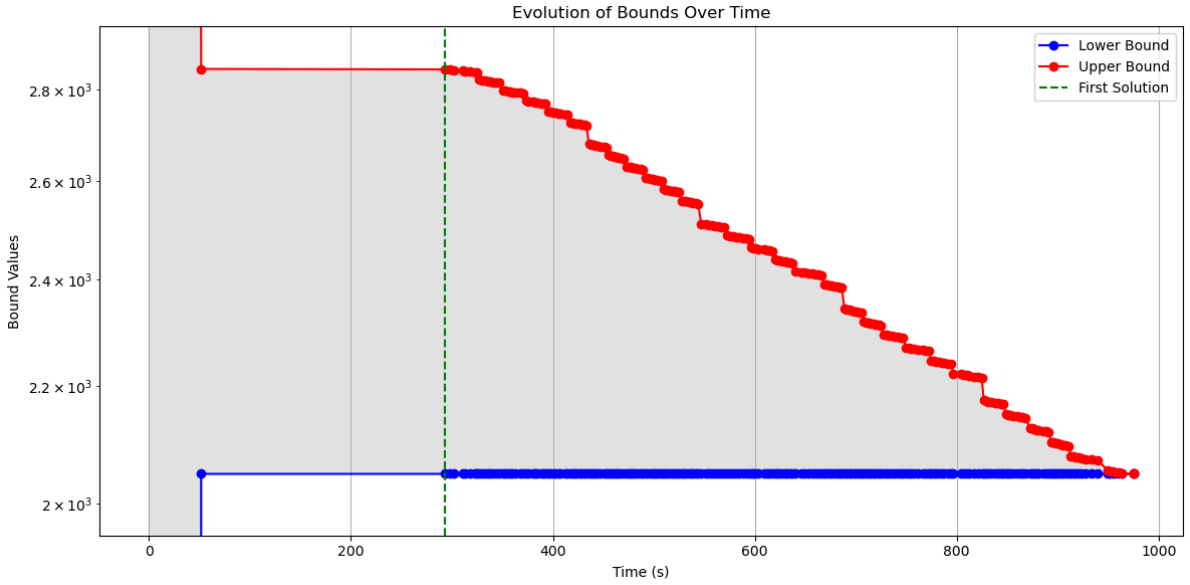
7.1.1 Costi

Per rimanere il più fedeli possibili ad uno scenario reale, tutti i test che seguono sono stati eseguiti utilizzando lo stato macchine che ci è stato fornito da Lemur della settimana tra il 25 e il 30 Novembre. Inoltre, gli ordini richiesti sono generati utilizzando il generatore di dati sintetici che ritorna una situazione verosimile sulla base degli ordini presenti nello storico.

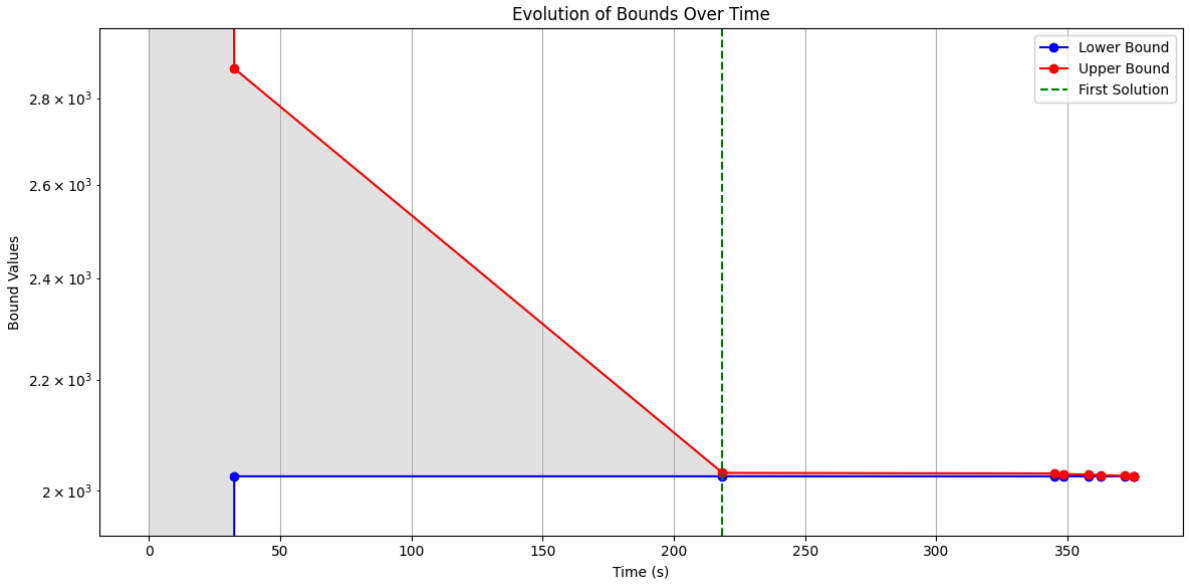
Makespan Minimization

Di seguito possiamo vedere i grafici che otteniamo andando a considerare i dati di limite superiore ed inferiore per le soluzioni trovate dallo scheduler. Il solver è in grado di trovare una soluzione valida in tempi ragionevoli e, se lasciato eseguire per più tempo, riesce a raffinare la soluzione fino a trovare quella ottimale, anche per problemi di dimensione non irrisoria.

Per generare il grafico seguente abbiamo utilizzato uno stato macchine reale combinandolo con 21 nuovi ordini per una richiesta complessiva di 23'674.53 Kg di prodotti vari con un orizzonte temporale pari a 120 giorni. Usando una macchina con un processore AMD 5700U siamo in grado di ritornare una soluzione valida dopo circa 5 minuti di attesa.

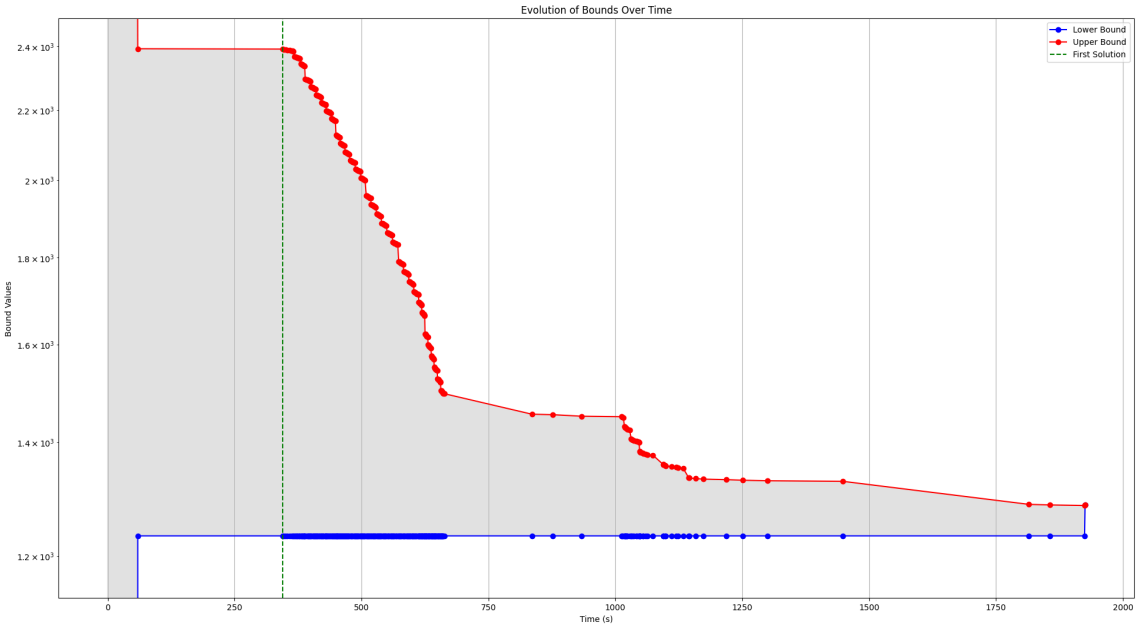


Una nota interessante è da fare sull'esecuzione su diverse macchine (computer), in quanto utilizzando un processore più potente con soli 8 thread (processori logici) in più siamo stati in grado di ottenere il grafico di seguito adottando minimi ritocchi al primo stadio di ricerca, permettendoci di riprodurre risultati consistenti in diverse esecuzioni consecutive:



Questo indica che la nostra soluzione ha la possibilità di scalare bene con configurazioni hardware più performanti. In particolare: utilizzare processori (CPU) più moderni permette al solver di eseguire un maggior numero di ricerche in parallelo, consentendogli di adottare internamente strategie diverse che possono notevolmente ridurre il tempo di esecuzione.

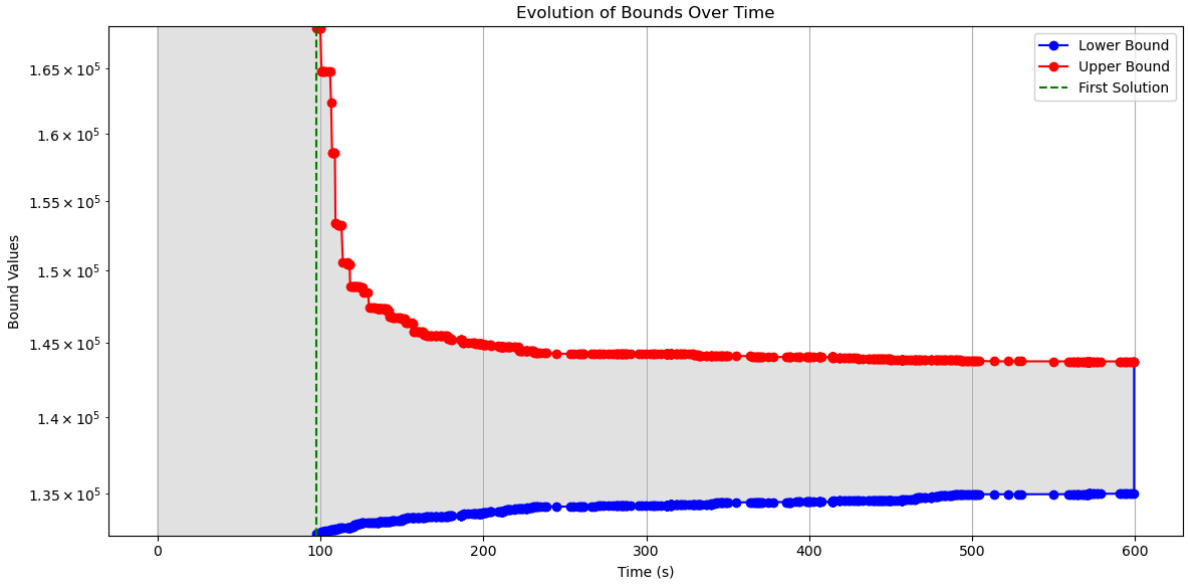
Abbiamo anche voluto estendere la ricerca ad un problema più grande per studiare come la nostra soluzione scalasse con l'aumentare della complessità. Di seguito è riportato il grafico dei limiti ottenuti dalla ricerca del solver:



In questo caso, il solver è ancora in grado di trovare una soluzione valida in tempi accettabili (circa 6 minuti) ed ottimizzarla in seguito arrivando ad un buon risultato dopo 20 minuti (1200 secondi nel grafico). Ma l'ottimale diventa complesso da trovare data la natura del problema.

Compactness Maximization

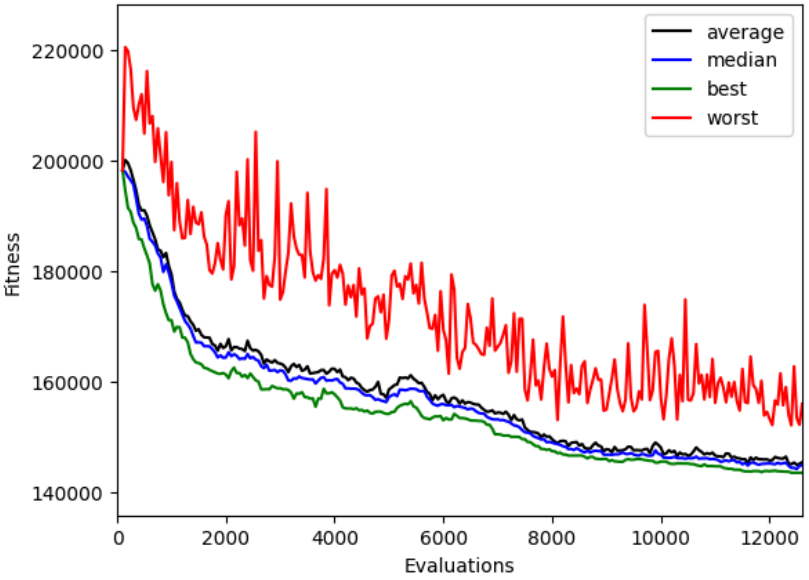
Spostando la nostra attenzione al secondo step di ottimizzazione e tornando al problema originale con 21 prodotti: si nota come siamo in grado migliorare di molto la soluzione iniziale in poco tempo (sempre attorno ai 5 minuti).



Purtroppo, trovare poi l'ottimo richiede molto più tempo, dovendo ricercare uno spazio delle soluzioni molto più ampio. E' anche questo il motivo per cui abbiamo deciso di sfruttare un approccio genetico per migliorare ulteriormente questa soluzione.

Genetic Refinement

Questa componente è in grado di ottimizzare la soluzione trovata dagli step precedenti nella maggior parte dei casi. Volendo mostrare le piene potenzialità di questa componente genetica, mostriamo di seguito le performance riscontrate utilizzando come input direttamente la soluzione trovata dal primo stadio di ottimizzazione, omettendo dunque il secondo.



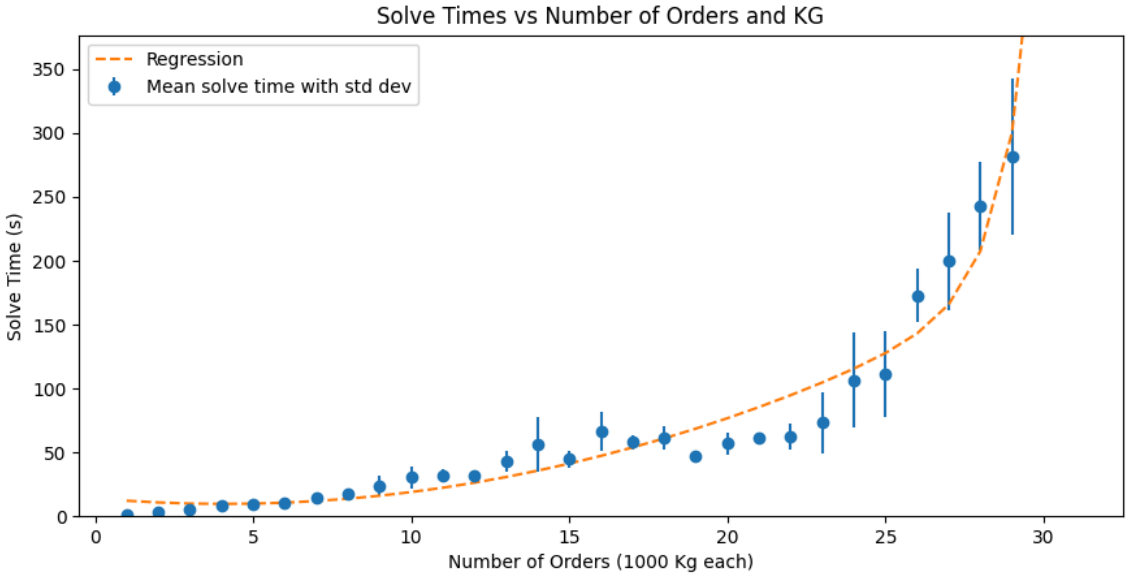
Come si può vedere dal grafico, introducendo mutazioni per scambiare e compattare i cicli, la ricerca genetica è in grado di trovare una soluzione molto vicina a quella del secondo step di ricerca in tempi di esecuzione notevolmente ridotti, per il grafico riportato abbiamo utilizzato 250 generazioni che impiegano meno di 3 minuti per eseguire!
Nonostante ciò, noi crediamo che sia comunque utile mantenere il secondo stadio di ottimizzazione, questo poiché fornisce maggiori garanzie nel trovare una soluzione che rispetta tutti i requisiti, cosa che la componente genetica non sempre rispetta. Da quanto riscontrato in fase di testing, raramente accade che questa componente restituisca una soluzione non valida. Nel caso in cui avvenga, verrà sempre utilizzata la soluzione fornita dallo step precedente.

Nel complesso: considerato il basso costo in termini temporali che ha lo stadio finale di ottimizzazione genetica, nonché le sue potenzialità di ricerca ad-hoc studiate sul problema di LeMur (che il secondo stadio in parte trascura), il nostro consiglio è mantenere tutte e 3 le fasi di ottimizzazione. Questo ci permette di definire una pipeline completa, la quale assicura sia validità che efficacia delle soluzioni.

Valid Solution Time

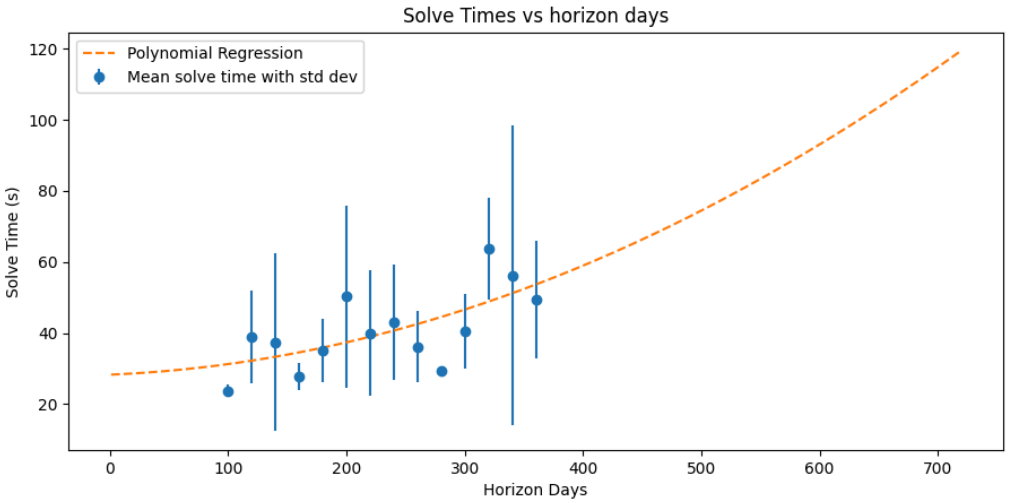
Estrapolando dalle analisi precedenti, trovare la soluzione più ottimale non è sempre possibile in tempi utili. Nonostante ciò, abbiamo riscontrato che una volta trovata una soluzione valida si riesce in poco tempo (specialmente tramite il genetic refinement) ad ottimizzare tale soluzione.
Un parametro molto più importante per noi è quindi in quanto tempo siamo in grado di trovare una soluzione valida a partire dal primo step risolutivo. Per studiare questo comportamento, abbiamo usato una funzione che ci ha permesso di raccogliere il costo della ricerca con diverse condizioni.

Il primo studio si concentra sulla relazione tra il numero di ordini e il numero di Kg richiesti come nuovi ordini (considerando uno stato macchine vuoto). I risultati sono riportati nel grafico di seguito:



Da questo possiamo osservare che all'aumentare della dimensione del problema, intuitivamente, aumenta anche il tempo necessario al solver per trovare la prima soluzione, in maniera super-lineare. Più interessante è come anche la deviazione standard, calcolata su 5 esecuzioni del software a pari condizioni aumenti con il problema. Questo comporta che con problemi molto grandi si potrebbe incorrere in alcuni casi in cui il solver non è in grado di trovare una soluzione, ma allo stesso tempo se ri-eseguito questo potrebbe riuscire a trovarla in poco tempo. Purtroppo, questa è una limitazione intrinseca allo strumento che abbiamo deciso di utilizzare e alle tecniche usate per risolvere questo tipo di problemi. Nonostante ciò, rimane un risultato consono e più che soddisfacente alle esigenze poste ed il tempo a disposizione.

Un secondo studio è stato condotto sull'impatto dell'horizon utilizzato per il solver, ovvero il numero di giorni totali su cui è richiesta la ricerca di una soluzione. In questo caso abbiamo deciso di andare a vedere quanto questo impattasse sul trovare la soluzione ottimale al problema. I risultati sono riportati nel grafico di seguito:



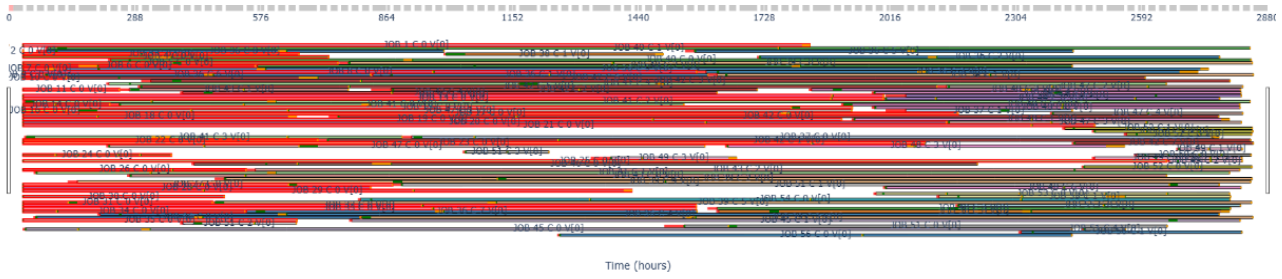
Come si può vedere la regressione riporta come questo parametro sembri parzialmente impattare i tempi. Ma al contempo i dati raccolti mostrano come la variabilità è considerevole, dovuta al problema discusso nel punto precedente.

Complessivamente, le performance raggiunte sono molto soddisfacenti, mantenendo bassi tempi di ricerca pure per orizzonti di programmazione ampi e richieste di produzione realistiche e considerevoli.

7.1.2 Solution Refinements

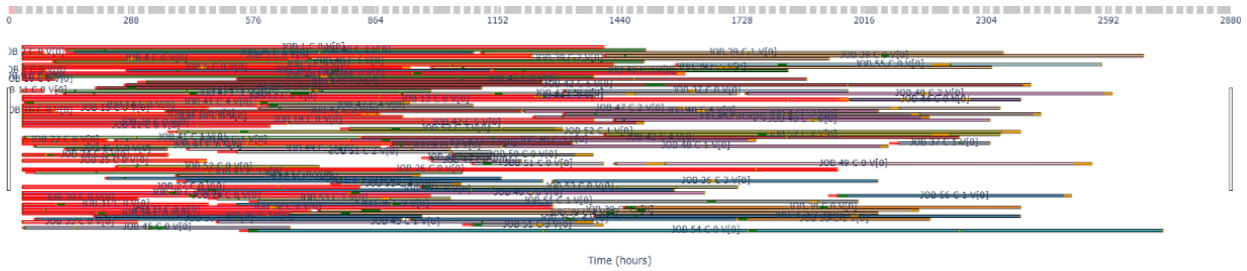
Andando ad osservare le soluzioni ritornate dai vari step, possiamo vedere la progressione della soluzione e come questa venga resa più compatta. Il problema usato è lo stesso introdotto inizialmente con 35 macchine in esecuzione e 21 ordini in entrata.

Il risultato del primo stadio di ottimizzazione è il seguente:



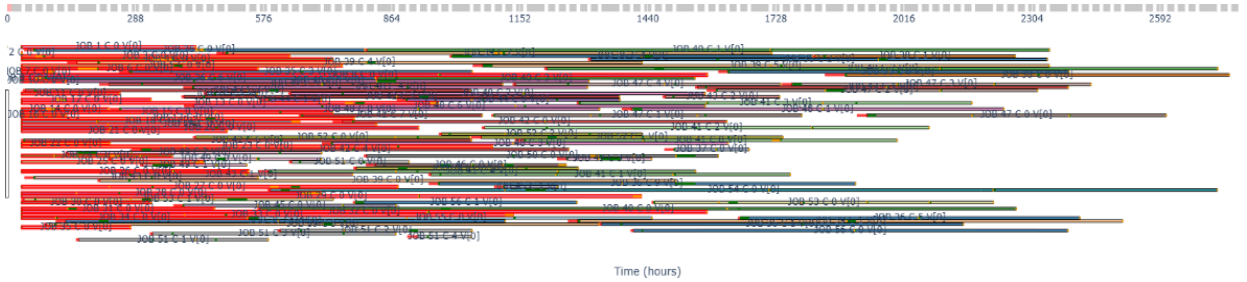
In tale fase si ottimizza solo il “caso peggiore”, ignorando il resto dei cicli. Questo risulta in 9390.75 giorni cumulativi (sommando la fine di ogni ciclo come da formula riportata nella sezione 4.3).

Per compattare i cicli si introduce il secondo stadio di ottimizzazione:



Il cui risultato porta la somma dei giorni cumulativi a 6353.75. E possiamo anche visivamente notare come la soluzione non sia più “spalmata” sull’intero makespan di produzione.

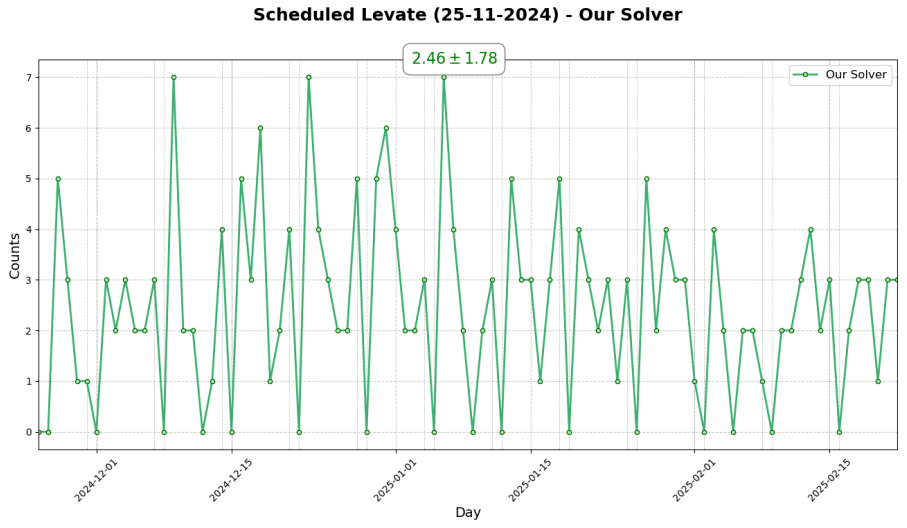
Per concludere, lo stadio di rifinimento genetico è in grado di ottimizzare ulteriormente la soluzione:



Lo scheduler ha quindi prodotto una soluzione con 5903.54 giorni cumulativi, questo implica un miglioramento di 3487.21 giorni sull’originale!

7.2 Confronto con pianificazioni reali

Per confrontare la pianificazione generata dal nostro sistema e quella attualmente adottata da LeMur abbiamo deciso di utilizzare il numero di levate giornaliere. Nel nostro approccio, il conteggio delle levate è stato effettuato con un’attenzione particolare alla struttura del processo produttivo, considerando tutte le fasi operative (setup, load, running, unload). Questo metodo permette di ottenere una rappresentazione più accurata e realistica del carico di lavoro quotidiano, basandosi sui dati riferiti alla giornata del 25 novembre 2024.



Uno degli aspetti più rilevanti emersi dall'analisi è la stabilità delle statistiche prodotte dal nostro sistema. La media giornaliera delle levate si è dimostrata pienamente coerente con il numero di operatori disponibili, configurati in input come 2 gruppi da 4 operatori ciascuno. Questo dimostra la capacità del nostro scheduler di modellare il carico di lavoro in modo equilibrato, evitando carichi troppo elevati o ridotti assegnati alle risorse umane.
Inoltre, la varianza associata alle levate giornaliere risulta contenuta, evidenziando una distribuzione uniforme degli interventi manuali e riducendo i picchi di attività che potrebbero causare inefficienze o difficoltà operative. Interessante notare che l'attenta modellazione del problema da noi eseguita ha permesso l'emergere di questa proprietà "almost for free", non avendo modellato in nessun modo esplicito una penalità per soluzioni sbilanciate.

La pianificazione attuale di LeMur, rappresentata principalmente tramite grafici di Gantt, tende a semplificare la gestione di queste operazioni, trascurando la disponibilità degli operatori e i vincoli temporali.
Da un'analisi preliminare sui dati interpretati dai Gantt fornitoci da LeMur, è emerso come queste possano sottostimare il carico operativo, aumentando il rischio di sovraccarico per il personale.

L'aspetto più significativo emerso è quindi la capacità del nostro sistema di adattarsi alla forza lavoro disponibile. Questa caratteristica consente non solo di aumentare l'efficienza operativa, ma anche di migliorare l'ambiente di lavoro, riducendo il rischio di stress legato a picchi di attività concentrati in determinati momenti della giornata.

7.3 Validazione dei risultati

Ognuno dei risultati precedentemente elencati è stato valutato direttamente dai rappresentanti LeMur. Hanno confermato la qualità del programma e dei risultati ottenuti, anche aiutandoci durante il processo di sviluppo e testing del software dando una prospettiva più pratica e realistica.

8. Integrazioni e contenuti aggiuntivi

Per facilitare l'interazione con il nostro sistema, nonché l'integrazione di questo all'interno dell'attuale gestionale di LeMur, abbiamo provveduto all'implementazione di componenti aggiuntive, determinanti per un facile utilizzo del nostro applicativo.

- **Applicazione Web**: Un'interfaccia web locale permette di interagire con gli input del sistema con maggiore facilità, nonché di modificare dei parametri senza accedere al codice. Inoltre, è possibile visionare ed interagire attivamente con il grafico di Gantt risultante dalla ricerca, facilitando così anche eventuali operazioni di analisi o comprensione dell'output.
  Un video illustrativo della web app è presente a questo link.

- **Convertitore**: Una componente che utilizza in input le tabelle di Pianificazione e Stato macchine (correttamente provviste di codici interni), restituendo in output l'equivalente stato attuale della linea produttiva in un formato compatibile con quello atteso dal nostro software. Questo elemento ci ha permesso di estrarre con facilità i dati utilizzati nelle fasi finali di benchmark dell'applicativo, inoltre, può essere un ottimo valore aggiunto per LeMur, in quanto semplifica ulteriormente l'interazione con il sistema, andando a creare autonomamente i file "*new_orders.csv"* e "*running_orders.csv"* specificati in (5.1). Ulteriori test sarebbero opportuni per confermare la perfetta funzionalità della componente, ma risulta indubbiamente essere uno strumento utile

9. Impatto sull'azienda

Adottare un sistema simile può comportare una serie di notevoli benefici a LeMur, tra cui:

1. **Aumento dell'efficienza produttiva:** in termini di Kg. giornalieri, dovuto all'introduzione di un sistema software complesso e meno sensibile agli errori umani.
2. **Maggiore orizzonte di pianificazione**: grazie alla possibilità di programmare nel lungo futuro (1 anno ad esempio) la pianificazione da seguire.
3. **Migliore ambiente lavorativo**: dovuto a una migliore distribuzione giornaliera degli interventi manuali.

In aggiunta, un quarto punto, decisamente poco tecnico ma altrettanto valido dal nostro punto di vista si può riassumere in "**Accogliere il cambiamento"**. Crediamo che dare atto a LeMur delle potenzialità di sistemi come questo, anche concepiti da zero nell'arco di soli 2 mesi, sia un più che valido incentivo nel potenziare la propria infrastruttura informativa, adottando ad esempio sistemi gestionali più moderni e strutturati, i quali permetterebbero di integrare con maggiore facilità strumenti informatici come quello da noi proposto.

10. Direzioni future e Limitazioni

In un'ottica di estendere il progetto, proponiamo una serie di funzionalità mancanti, potenzialmente benefiche:

- Possibilità di suggerire **produzioni "fantasma"** (in assenza di un ordine ricevuto) per alcuni articoli. Idealmente, considerato un certo istante temporale, sarebbe ottimale poter predire quali siano gli articoli con maggiore probabilità di essere richiesti nel futuro prossimo, con relativo quantitativo in Kg. Tale sistema potrebbe essere implementato tramite soluzioni di machine learning (e.g. recommender systems) ed estenderebbe l'input del nostro scheduler. Una componente come questa permetterebbe di ridurre i tempi di inattività della catena produttiva.
- Gestione della **velocità** e del consumo energetico delle macchine. Si noti che questa funzionalità è già parzialmente implementata nello scheduler presentato, ma non ancora integrata nella funzione obiettivo del sistema. Questi vincoli permetterebbero di velocizzare o rallentare la produzione, così da completare più ordini possibili durante i periodi di grande richiesta, al contrario, di minimizzare il consumo energetico e la capacità produttiva nei momenti in cui la domanda sia contenuta.
- Creazione di uno strumento che automatizzi l'integrazione con il gestionale LeMur, in modo tale da snellire il passaggio fra l'output dello scheduler e l'input nel sistema LeMur. Come accennato in (9) a nostro avviso rimane comunque una scelta migliore aggiornare prima il gestionale stesso.

A. Link utili

- *Miro Board Team 3* - https://miro.com/app/board/uXjVLc7ULgY=/

B. Contatti

- Solvers:
  - Alessandro Lorenzi
  - Andrea De Carlo
  - Davide Cavicchini
  - Emanuele Poiana
  - Luca Cazzola
  - Luca Sperandio
  - Silvano Vento Maddonni
- Mentori:
  - Matteo Gerola
  - Matteo Zanoni
  - Federico Pederzolli
- Referente LeMur:
  - Pietro Modena

C. Referenze

[1] *OR-Tools v9.11. Laurent Perron and Vincent Furnon. https://developers.google.com/optimization/*

[2] *inspyred -- A framework for creating bio-inspired computational intelligence algorithms in Python https://pythonhosted.org/inspyred/reference.html*

[3] *Pinedo, M. L. (2016). Scheduling: Theory, algorithms, and systems (5th ed.). Springer.*

[4] *Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. Computers & Operations Research, 13(5), 533–549. https://doi.org/10.1016/0305-0548(86)90048-1*

[5] *Brucker, P. (2007). Scheduling algorithms (5th ed.). Springer.*

[6] *Hooker, J. N. (2000). Logic-based methods for optimization: Combining optimization and constraint satisfaction. Wiley.*

[7] *Héctor G.-de-Alba, Samuel Nucamendi-Guillén, Oliver Avalos-Rosales, A mixed integer formulation and an efficient metaheuristic for the unrelated parallel machine scheduling problem: Total tardiness minimization, EURO Journal on Computational Optimization, Volume 10, https://doi.org/10.1016/j.ejco.2022.100034.*

[8] *Mitsuo G. Solving job-shop scheduling problems by genetic algorithm. Proceedings of IEEE International Conference on Systems, Man and Cybernetics*