

UNIVERSITÀ DEGLI STUDI DI MILANO

Facoltà di Scienze e Tecnologie

*Corso di Laurea in Informatica Magistrale*



Modellazione e analisi di Task Scheduling  
in ambiente Green Edge Computing

*Relatore:* Christian Quadri

Tesi di Laurea Magistrale di:

Leonardo Menti

Matricola N. 982296

ANNO ACCADEMICO 2021/2022

# Indice

<b>Indice</b>	<b>i</b>
<b>Lista delle figure</b>	<b>iv</b>
<b>Introduzione</b>	<b>1</b>
<b>1 Related Works</b>	<b>5</b>
1.1 Task scheduling . . . . .	6
1.2 Architettura infrastrutturale . . . . .	7
1.3 Ottimizzazione energetica . . . . .	8
1.4 Applicazioni specifiche . . . . .	8
1.5 Novità del nostro lavoro . . . . .	9
<b>2 Modello</b>	<b>11</b>
2.1 Programmazione lineare intera . . . . .	12
2.2 Modello green edge . . . . .	13
2.2.1 Task . . . . .	15
2.2.2 Modello . . . . .	15
2.3 Modello green edge con accumulatori . . . . .	17
2.3.1 Modello . . . . .	20

<b>3 Implementazione</b>	<b>22</b>
3.1 File di configurazione . . . . .	23
3.2 Scenarios generator . . . . .	25
3.3 Execution engine . . . . .	27
3.4 Dati . . . . .	28
3.4.1 Dati sulla produzione energetica . . . . .	29
3.4.2 Dati sulla domanda energetica . . . . .	31
3.5 Topology . . . . .	33
3.6 Model . . . . .	35
3.6.1 Gurobi . . . . .	35
3.6.2 Implementazione del modulo . . . . .	35
3.7 Utils . . . . .	36
3.8 Standard System e Accumulator System . . . . .	36
3.9 Funzione di penalità . . . . .	36
<b>4 Euristica</b>	<b>37</b>
4.1 Euristica Standard . . . . .	38
4.1.1 Modello . . . . .	39
4.1.2 Implementazione . . . . .	40
4.2 Euristica con accumulatori . . . . .	42
4.2.1 Modello . . . . .	42
4.2.2 Implementazione . . . . .	44
<b>5 Analisi dei risultati</b>	<b>46</b>
5.1 Ambiente di test e generazione degli scenari . . . . .	47
5.2 Task accettati dal sistema . . . . .	49
5.3 Utile del sistema . . . . .	53
5.4 Saturazione del sistema nel tempo . . . . .	56

5.5	Euristica vs modello ottimo . . . . .	60
5.5.1	Utile . . . . .	60
5.5.2	Task accettati . . . . .	62
<b>Conclusioni</b>		<b>65</b>
<b>A File di configurazione</b>		<b>68</b>
<b>B Model</b>		<b>70</b>
B.1	Variabili Decisionali . . . . .	70
B.2	Vincoli . . . . .	71
B.3	Funzione Obiettivo . . . . .	74
<b>C Utils</b>		<b>75</b>
C.1	Generazione facilities . . . . .	75
C.2	Generazione task . . . . .	76
C.3	Import . . . . .	76
C.4	Export . . . . .	77
<b>Bibliografia</b>		<b>78</b>

# Elenco delle figure

1	Schema di alimentazione . . . . .	14
2	Schema di alimentazione con accumulatori . . . . .	19
3	Schema di implementazione . . . . .	23
4	Tipologie di task . . . . .	25
5	Esempio di scenario . . . . .	27
6	Database . . . . .	28
7	Topologia della rete . . . . .	29
8	Photovoltaic power production . . . . .	30
9	Potenza energetica giornaliera nella regione di Antwerp . . . . .	31
10	Market Load . . . . .	32
11	Rapporto tra domanda energetica e prezzo . . . . .	33
12	Belgian National Research and Education Network (BELNET) . . . . .	34
13	Schema di implementazione euristica . . . . .	41
14	Percentuale di task accettati su un totale di 1000 . . . . .	49
15	Percentuale di accettazione per tipologia di task su un totale di 1000	51
16	Percentuale di task accettati su un totale di 5000 . . . . .	52
17	Percentuale di accettazione per tipologia di task su un totale di 5000	53
18	Utile del sistema con 1000 task . . . . .	54

19	Utile del sistema con 5000 task . . . . .	55
20	Saturazione del sistema nel tempo con 1000 task . . . . .	56
21	Disponibilità green e prezzo dell'energia brown in West-Flanders . . . . .	57
22	Saturazione del sistema nel tempo con 5000 task . . . . .	59
23	Perdita percentuale dell'euristica con 1000 task . . . . .	61
24	Perdita percentuale dell'euristica con 5000 task . . . . .	61
25	Tasso di accettazione task dell'euristica con 1000 task . . . . .	63
26	Tasso di accettazione task dell'euristica con 5000 task . . . . .	63
27	Esempio di task . . . . .	76

# Introduzione

La comunicazione tra i dispositivi digitali è alla base del funzionamento, della gestione e più in generale delle dinamiche della società moderna. I dispositivi elettronici necessitano di una connessione ad Internet e richiedono la gestione dei dati da loro generati in modo da partecipare attivamente al funzionamento di un applicativo. Tuttavia è spesso richiesto un servizio più elaborato rispetto ad una classica applicazione general purpose, dove possono sorgere problematiche di latenza, sicurezza o scalabilità. Per contrastare queste problematiche viene introdotto il paradigma *Edge Computing*, una tecnologia che permette di eseguire elaborazioni il più vicino possibile ai dispositivi che generano i dati, riducendo la latenza di comunicazione e il traffico di rete. Applicazioni come sistemi di realtà virtuale, di guida autonoma o di *Internet of Things (IoT)* sono alcuni esempi in cui è richiesta una bassa latenza per funzionare efficacemente, in quanto il tempo trasmesso per comunicare con il cloud può essere troppo elevato, rendendo l'applicazione inutilizzabile. Come anticipato, l'Edge Computing è una risorsa importante anche in ambito di sicurezza dei dati. Un traffico diretto al cloud è sicuramente più vulnerabile ad attacchi informatici, mentre la comunicazione con l'edge della rete, essendo fisicamente vicino al dispositivo, riduce il rischio di violazioni della sicurezza. L'Edge Computing consente inoltre di gestire una grande quantità di dati generati dai dispositivi e di scalare facilmente l'infrastruttura di rete.

Tuttavia il deployment di applicazioni nell'Edge Computing presenta alcuna sfide, come la distribuzione e l'utilizzo delle risorse computazionali o l'ottimizzazione del consumo energetico. In questo contesto la schedulazione dei task nell'infrastruttura edge diventa un fattore critico per garantire un funzionamento efficiente e sostenibile del sistema. La schedulazione dei task (in inglese *task scheduling*) consiste nell'assegnare i task, o porzioni di questi, ai nodi dell'infrastruttura edge in ottica di essere processati. La schedulazione dei task mira a ottimizzare certi obiettivi prefissati del sistema, come la minimizzazione del tempo di esecuzione, la riduzione del consumo energetico o la massimizzazione del guadagno da parte del provider di rete. La complessità di questo problema dipende da un diverso numero di fattori, come la topologia della rete, il numero dei nodi, il numero dei task, la disponibilità energetica oppure la capacità di elaborazione. Tutti questi fattori rendono il problema di *task scheduling* difficile da risolvere in maniera ottimale.

Il discorso si può estendere considerando la così detta infrastruttura di *Green Edge Computing*. In questo ambiente si assume che i nodi dell'infrastruttura edge siano alimentati anche da fonti di energia rinnovabili, oltre alle classiche fonti derivate da combustibili fossili come petrolio, carbone e gas naturale. I nodi dell'infrastruttura sono invece alimentati anche da fonti come energia solare, eolica e/o idroelettrica. L'infrastruttura di Green Edge Computing rappresenta una soluzione innovativa ed ecologica per il settore delle Tecnologie dell'Informazione e della Comunicazione. In uno scenario del genere il gestore dell'infrastruttura può progettare la schedulazione dei task in modo da minimizzare l'impatto ambientale e allo stesso tempo massimizzare il guadagno derivante dal processamento dei task, considerando che la produzione di energia rinnovabile ha un costo nettamente inferiore rispetto all'energia non rinnovabile. In un'era in cui la sensibilizzazione ambientale è diventata una priorità globale, l'adozione di soluzioni tecnologiche sostenibili, come l'infrastruttura di *Green Edge Computing*, rappresenta un passo fondamentale verso una società più eco-sostenibile.

In questo lavoro di tesi è stato studiato e implementato un modello di schedulazione di task in ambiente green edge computing, con l'obiettivo di massimizzare il guadagno del provider dell'infrastruttura edge. Attraverso l'impiego della programmazione lineare intera si è formalizzato il problema, producendo due varianti del modello. La prima è caratterizzata da nodi di rete che sono alimentati da fonti energetiche rinnovabili e non rinnovabili. La seconda versione, concepita come estensione della prima, consente inoltre ai nodi di immagazzinare energia proveniente da fonti rinnovabili per un utilizzo futuro. Ciò ha portato ad un arricchimento della descrizione del sistema, consentendo ad ogni nodo di disporre di una fonte energetica aggiuntiva. Il modello, in entrambe le sue soluzioni, rappresenta una soluzione centralizzata, in cui un unico decisore è responsabile della schedulazione dei task. Il modello ha la visione totale degli avvenimenti nel tempo nell'infrastruttura, riuscendo quindi a risolvere in maniera ottimale il problema e restituire un *upper bound* per un eventuale algoritmo risolutivo non ottimale.

Sulla base del modello di programmazione lineare intera, è stata sviluppata una prima versione di un algoritmo risolutivo per gestire il problema di scheduling dei task in ambiente green edge computing. L'algoritmo da noi ideato combina tecniche euristiche e di algoritmi di programmazione lineare e fornisce un contributo importante per il problema affrontato in questa tesi. L'algoritmo è anche in questo caso di tipo centralizzato e presenta quindi un unico decisore che alloca i task nel sistema. Ovviamente il decisore non possiede la visione totale degli avvenimenti, ma gestisce i task nel loro ordine di arrivo.

Infine è stata condotta una campagna sperimentale configurando diversi scenari, utilizzando dati pubblici per la creazione della topologia di rete e la produzione di energia green. In virtù dei test eseguiti abbiamo analizzato i risultati ottenuti, valutando le prestazioni dei modelli di programmazione lineare intera e dell'euristica. In particolare abbiamo esaminato fattori come il guadagno del sistema, il numero e

tipologia di task accettati e la saturazione energetica dell’infrastruttura, confrontando infine il rendimento del modello e dell’euristica. I risultati ottenuti sono incoraggianti e dimostrano l’efficacia dell’algoritmo da noi proposto, sia in termini di guadagno del sistema che di numero di task accettati.

**Organizzazione della tesi** Il resto della tesi è organizzata come segue. Nel Capitolo 1 forniamo una breve overview dei lavori presenti in letteratura che affrontano, nella totalità o in parte, i concetti di task scheduling in diversi ambienti di rete. Il modello di programmazione lineare viene formalizzato nel Capitolo 2 e la sua implementazione è descritta nel Capitolo 3, dove vengono analizzate le tecnologie utilizzate e le scelte implementative. Il Capitolo 4 presenta la modellazione e la realizzazione dell’euristica da noi proposta per affrontare il problema di task scheduling in ambiente green edge. I risultati delle analisi condotte sul modello e sull’euristica sono presentati nel Capitolo 5.

# Capitolo 1

## Related Works

In questo capitolo vengono illustrati i così detti *related works*, ossia i lavori presenti in letteratura correlati al nostro progetto, che hanno in qualche modo ispirato ed orientato questa tesi di laurea. Verranno quindi elencati e discussi altri studi o pubblicazioni pertinenti al tema di ricerca di *Green Edge Computing* e *Tasks Scheduling*. I due ambiti stanno acquisendo sempre maggiore importanza accademica e di ricerca negli ultimi anni, in quanto consentono di affrontare le sfide di ottimizzazione energetica e computazionale nei sistemi informatici distribuiti. L'obiettivo principale della ricerca in questo campo è quello di sviluppare soluzioni software e hardware che possano consentire alle infrastrutture di rete di eseguire i tasks di dispositivi mobili e IoT con un consumo energetico ridotto, senza compromettere le prestazioni del servizio. La ricerca in questo ambito può avere un impatto significativo sull'industria e sulla società nel suo complesso, rappresentando quindi una sfida importante e stimolante per il mondo informatico. Di seguito una panoramica sui *related works*, con un'enfasi particolare sugli ambiti di task scheduling, architettura infrastrutturale, ottimizzazione energetica e applicazioni specifiche. Infine una descrizione con le principali differenze che il lavoro di questa tesi evidenzia rispetto alla letteratura.

## 1.1 Task scheduling

L’ambito di task scheduling, ovvero la pianificazione delle attività da eseguire in un sistema, è un problema fondamentale in molti ambiti di ricerca, tra cui l’informatica, l’ingegneria e la produzione industriale. L’obiettivo del task scheduling è quello di garantire un’efficienza energetica ottimale del sistema, riducendo al minimo il consumo di energia e massimizzando le prestazioni. In questo contesto, numerosi approcci di task scheduling sono stati proposti in letteratura, spaziando dalle tecniche basate su algoritmi di ottimizzazione, a quelle basate su approcci di machine learning e deep learning. In particolare in [1] viene proposta una soluzione di task offloading per applicazioni di gioco in ambiente edge computing, sviluppando successivamente un algoritmo intelligente di allocazione e migrazione delle sessioni di gioco online sotto diversi vincoli realistici. In [2] viene analizzata una soluzione di scheduling e offloading in un ambiente multi-livello green edge. L’approccio proposto punta a minimizzare il costo di esecuzione dei task, tenendo conto dei costi del sistema che includono latenza, consumo energetico e costi di utilizzo del cloud. In questo lavoro viene successivamente presentato un algoritmo di risoluzione online, che integra la tecnica di ottimizzazione di Lyapunov [3] e un efficiente algoritmo di arrotondamento dell’ottimizzazione. In [4] viene invece proposta una soluzione differente rispetto alle tradizionali euristiche, ossia un algoritmo di job scheduling basato su deep reinforcement learning esteso in un ambiente di multiple server clusters. Nonostante questo lavoro non sia presentato direttamente in ottica green edge computing, può risultare di buona ispirazione e facilmente integrabile nel contesto. Lo studio rivela che il metodo di deep reinforcement learning ha il potenziale per superare i tradizionali algoritmi di allocazione delle risorse in una varietà di ambienti complicati.

## 1.2 Architettura infrastrutturale

L’infrastruttura di rete è costituita generalmente dall’insieme di dispositivi hardware e software che permettono la comunicazione tra i vari nodi connessi alla rete. L’architettura delle infrastrutture è fondamentale per garantire la connettività tra gli utenti, ma anche per garantire sicurezza e qualità del servizio. I lavori citati nella precedente sezione 1.1 sono calati in architetture di rete diverse tra di loro, influenzando di conseguenza le rispettive rappresentazioni del problema e le successive implementazioni. In particolare in [1] viene modellata un’infrastruttura di rete edge 5G a più livelli [5] contenente una serie di edge game server. Vengono considerati due tipi di server, che differiscono per capacità e vicinanza agli utenti finali. Alcuni server risiedono su nodi remoti, ciascuno distribuito su una base station (BS) e destinato principalmente a servire gli utenti di quella BS. Il resto dei server si trova ciascuno su nodi posizionati nella rete edge/trasporto dove converge il traffico di più BS. Questi nodi hanno capacità maggiori rispetto ai nodi edge, in termini di potenza di calcolo, energia e memoria, sebbene queste capacità debbano essere condivise tra utenti appartenenti a diverse celle. Al contrario l’architettura presentata in [2] è più complessa e presenta diversi livelli dell’infrastruttura edge. La rete proposta prevede un set di server chiamato *Front Edge Tier (FET)* in prossimità agli utenti finali. Questi server offrono accesso a bassa latenza e sono dotati di un modulo per raccogliere energia green e immagazzinarla nelle loro batterie. L’architettura prevede un secondo set di server, il *Back-end Edge Tier (BET)*, posizionato più internamente nella rete e caratterizzato da maggior potenza computazionale e disponibilità energetica. Infine è previsto anche il livello cloud che comunica con il resto dell’infrastruttura e processa carico di lavoro all’occorrenza. Lo studio si concentra sul consumo energetico dei server edge (sia in FET che in BET), sui costi di esecuzione del cloud e sulla latenza di comunicazione all’interno dei server edge e nei server cloud.

### 1.3 Ottimizzazione energetica

Il problema dell’ottimizzazione energetica è un tema importante nell’ambito della ricerca e si concentra sull’utilizzo efficiente e sostenibile dell’energia, riducendo i costi e l’impatto ambientale delle attività umane. Negli elaborati proposti in precedenza il principale obiettivo è quello di massimizzare il guadagno generato dal processamento dei task. Tuttavia operando all’interno di un’architettura green edge le soluzioni proposte dai diversi studi portano diversi vantaggi. Per prima cosa il green edge computing utilizza hardware e software efficienti dal punto di vista energetico, con una conseguente riduzione dell’impatto ambientale. In secondo luogo l’utilizzo di nodi green può ridurre i costi energetici, in quanto generalmente la produzione di energia rinnovabile ha un costo minore rispetto alla produzione di energia mediante combustibili fossili. Questo aspetto può risultare un grosso vantaggio dalla parte del provider di rete, che può generare un guadagno maggiore dovuto alle minori spese di alimentazione energetica.

### 1.4 Applicazioni specifiche

Lo studio del problema di task scheduling in ambiente green edge computing può avere applicazioni e ripercussioni in diversi ambiti. L’ottimizzazione delle risorse energetiche è una preoccupazione comune in molti settori, in quanto l’energia è una risorsa limitata e costosa e l’impatto ambientale delle attività informatiche può essere significativo. In [1] viene affrontato il problema di come supportare il gaming ai margini della rete internet. L’articolo, ricordando che l’edge computing gode rispetto al cloud di una latenza ridotta e di una maggiore larghezza di banda, evidenzia che un trasferimento di giochi basati sul cloud ad un livello edge può rappresentare un servizio premium per l’utente finale. L’obiettivo del lavoro è quello di progettare uno

schema compatibile con l’architettura MEC [6] e 5G, che massimizzi il guadagno di un provider di servizi/infrastrutture con capacità dei nodi edge variabili nel tempo grazie all’accesso a energia rinnovabile intermittente. In [7] invece gli autori si concentrano su applicazioni MEC per l’IoT e affrontano i temi di efficienza energetica e performance di offloading. Nell’articolo è possibile esaminare le diverse tipologie di applicazioni che un’infrastruttura edge può sostenere in ambito IoT, come il trasporto, l’agricoltura, la sanità o gli edifici intelligenti. Il fine del lavoro è quello di ottimizzare il guadagno dei fornitori di servizi, riducendo al contempo il costo energetico e il tempo di esecuzione delle attività dei dispositivi IoT. In [2] il focus applicativo è più generale e il carico di lavoro dei nodi edge è interessato da un più comune processamento di task. Tuttavia questo aspetto permette al lavoro di essere utilizzato in diversi ambiti, personalizzando e calando l’implementazione in scenari più specifici e particolari. In conclusione l’infrastruttura di rete di tipo green edge computing può essere utilizzata per supportare diverse applicazioni ed essere quindi utile in molti settori della società moderna. Lo scheduling dei task all’interno di tale architettura è di fondamentale importanza in quanto si concentra sull’ottimizzazione delle risorse disponibili, garantendo un utilizzo efficiente dell’energia e una riduzione dell’impatto ambientale.

## 1.5 Novità del nostro lavoro

Il prodotto di questa tesi è stato certamente ispirato e condizionato da tutti i riferimenti citati nelle sezioni precedenti. Il lavoro compiuto consiste nella creazione di un modello coerente al problema di task scheduling in un ambiente di green edge computing. I task generati dai vari nodi dell’infrastruttura non vengono semplicemente inviati da un nodo ad un altro per essere processati. Al contrario i task possono essere suddivisi in diverse porzioni che vengono gestiti indipendentemente dal sistema.

Le porzioni di task possono essere prese in carico da diversi nodi edge e processati in diversi momenti, in quanto assumiamo che le varie porzioni siano tra loro indipendenti. L’obiettivo del modello è quello di massimizzare il guadagno del gestore dell’infrastruttura, derivante dal processamento dei task. L’architettura ricreata è generica e presenta un set di server tutti allo stesso livello infrastrutturale. I nodi che compongono l’infrastruttura generano i vari task e allo stesso tempo sono potenziali candidati per il processamento di altri task. I nodi del sistema godono di accesso a energia rinnovabile variabile nel tempo, disponendo in ogni caso anche di fonti di alimentazione energetica non rinnovabile. Il modello creato è general purpose e la sua implementazione può essere riutilizzata in diversi ambiti, permettendo la modifica di molti parametri di configurazione. Il modello, dopo essere stato implementato e testato, ha prodotto una serie di risultati che ne provano l’efficacia, mostrando il diverso comportamento del sistema a seconda degli scenari che deve ottimizzare. Il modello creato in questo lavoro di tesi fornisce un upper bound per successivi algoritmi di risoluzione. Una prima versione di euristica risolutiva è stata creata, implementata e confrontata con il risultato ottimo prodotto dal modello.

## Capitolo 2

### Modello

Nel seguente capitolo viene descritto ed analizzato il modello di programmazione lineare ideato per risolvere il problema di task scheduling in ambiente di green edge clusters. Il problema affrontato in questa tesi è quello di assegnare porzioni di task all'interno di un'infrastruttura di facility. I task sono generati nella medesima infrastruttura di rete, per poi essere assegnati ad alcuni nodi per essere eseguiti, tendenzialmente i più vicini al nodo generatore. Il concetto di esecuzione è generale: un nodo riceve una quantità di informazioni che rappresenta un task o una sua porzione, per poi processare i dati e restituire un output al nodo che ha generato quel determinato task. Ogni facility è alimentata da fonti energetiche green e/o brown, con un conseguente costo di computazione dipendente dalla fonte di energia. Inoltre ogni task prevede un reward nel caso in cui il sistema processi interamente l'input del task, entro una determinata deadline. L'obiettivo dell'infrastruttura di rete è quello di massimizzare il guadagno, definito come la somma dei reward dei task che accetta meno il costo computazionale derivato dal processamento degli stessi task. Per prima cosa è stato definito un modello di programmazione lineare che rappresenti il problema di task scheduling nell'infrastruttura.

## 2.1 Programmazione lineare intera

La programmazione lineare intera [8] (PLI oppure in inglese integer linear programming - ILP) è una classe di problemi di ottimizzazione combinatoria che prevede di trovare la soluzione ottima di una funzione obiettivo lineare soggetta a vincoli lineari e ad ulteriori restrizioni che richiedono che le variabili siano intere.

Il problema di programmazione lineare intera può essere formalmente descritto come segue:

$$\max \quad \sum_{j=1}^n c_j x_j \quad \text{s.t.} \quad \sum_{j=1}^n a_{ij} x_j \leq b_i, \quad i = 1, \dots, m \quad x_j \in \mathbb{Z}, \quad j = 1, \dots, n$$

dove  $c_j$ ,  $a_{ij}$  e  $b_i$  sono numeri reali, e  $x_j$  sono variabili intere.

La programmazione lineare intera permette di modellare molti problemi reali. Ad esempio, la pianificazione della produzione industriale, la progettazione di reti di trasporto, la gestione della catena di approvvigionamento e la pianificazione finanziaria sono solo alcuni esempi di problemi che possono essere modellati come problemi di programmazione lineare intera.

Il problema di programmazione lineare intera è un problema di ottimizzazione difficile, poiché richiede di esplorare tutte le possibili combinazioni di valori interi delle variabili. Per questo motivo, la programmazione lineare intera è spesso risolta con algoritmi di ottimizzazione dedicati.

Esistono diversi algoritmi per risolvere problemi di programmazione lineare intera. Uno dei più comuni è il branch and bound, che si basa sulla divisione del problema in sotto-problemi più piccoli e la risoluzione di ciascuno di questi sotto-problemi per trovare la soluzione ottima. Un altro algoritmo comune è il branch and cut, che prevede di aggiungere iterativamente nuovi vincoli per restringere lo spazio delle soluzioni possibili.

In sintesi, la programmazione lineare intera è un'area di ricerca attiva e importante, in grado di modellare molti problemi reali. Sebbene la risoluzione di tali problemi possa essere difficile, esistono algoritmi e software specializzati che possono aiutare a trovare la soluzione ottima in modo efficiente.

## 2.2 Modello green edge

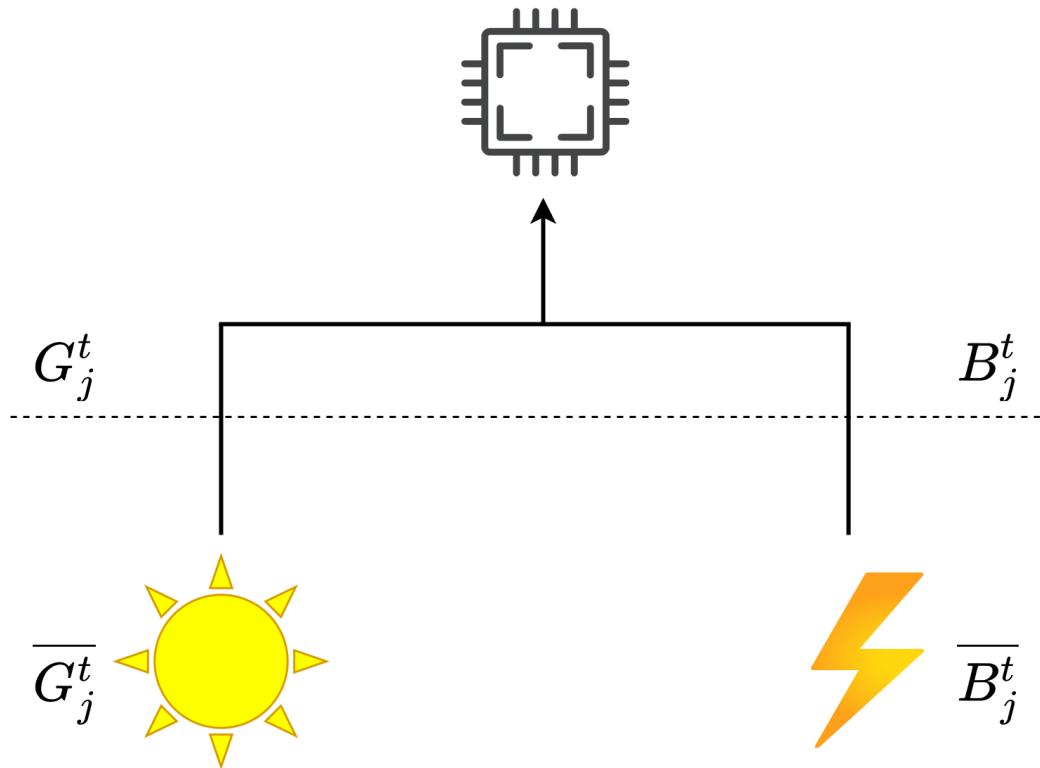
Nel nostro caso il problema di *task scheduling* può essere modellato con la programmazione lineare intera. Consideriamo un set di variabili di decisione binaria e tre set di variabili di decisione continue:

- $X_{ij}^t \in [0, 1]$ : rappresenta la porzione del task  $i$  assegnato alla facility  $j$  al tempo  $t$ .
- $Z_i \in \{0, 1\}$ : è una variabile che rappresenta l'accettazione e il conseguente processamento del task  $i$  entro la sua deadline.
- $G_j^t \in \mathbb{R}^+ \cup \{0\}$ : rappresenta le risorse computazionali, alimentate da *energia green*, allocate nella facility  $j$  al tempo  $t$  per processare i task del sistema.
- $B_j^t \in \mathbb{R}^+ \cup \{0\}$ : rappresenta le risorse computazionali, alimentate da *energia brown*, allocate nella facility  $j$  al tempo  $t$  per processare i task del sistema.

Assumiamo una rappresentazione discreta temporale utilizzando un insieme finito  $T$  di intervalli di tempo. Assumiamo inoltre che un insieme di  $F$  edge facility siano disponibili e che ognuna di esse fornisca risorse computazionali alimentate da energia green e brown. La quantità di risorse computazionali totale disponibile in una facility  $j$  è fissa nel tempo e dipendente dalla facility stessa. Supponiamo che in un determinato momento  $t$  una facility  $j$  abbia  $G_j^t \leq \overline{G}_j^t$  risorse di elaborazione disponibili

utilizzando energia green, mentre la capacità rimanente  $B_j^t$  può essere richiesta utilizzando l'energia brown. In Figura 1 è possibile visualizzare lo schema di alimentazione delle facility. In questo modello ogni facility dispone, in ogni timeslot, di  $\bar{G}_j^t$  energia green e  $\bar{B}_j^t$  energia brown. I nodi utilizzano queste due fonti per alimentare le loro CPU, rispettando il vincolo di capacità energetica.

Supponiamo uno scenario eterogeneo in cui ogni facility ha una capacità diversa ed una disponibilità diversa di risorse green e brown. Inoltre  $w_{gj}^t$  e  $w_{bj}^t$  sono il costo di una unità di capacità della struttura  $j$  al tempo  $t$  utilizzando l'energia green e brown, rispettivamente. In generale, assumiamo che  $w_{gj}^t \leq w_{bj}^t$ . Assumiamo il costo dell'energia green fisso nel tempo, mentre il costo dell'energia brown è variabile rispetto alla domanda registrate nel corso della giornata.



**Figura 1:** Schema di alimentazione

### 2.2.1 Task

Ogni task  $i$  è definito come segue:  $\langle a_i, \tau_i, d_i, o_i, \mu_i, r_i, k_i \rangle$ , dove:

- $a_i$  è l'orario di arrivo del task
- $\tau_i$  è la deadline del task
- $d_i$  rappresenta la dimensione dei dati in input (ad esempio in byte) dell'attività
- $o_i$  è la dimensione dell'output
- $\mu_i$  è la risorsa di CPU necessaria per elaborare un'unità di dati in input (ad esempio il ciclo di CPU)
- $r_i$  è il guadagno che il provider di rete otterrà se completa il task entro la sua deadline
- $k_i$  è la facility che genera il task

### 2.2.2 Modello

Di seguito il modello di PLI che rappresenta il problema:

$$\max \sum_{i \in N} r_i \cdot Z_i - \sum_{t \in T} \sum_{j \in F} (w_{gj}^t \cdot G_j^t + w_{bj}^t \cdot B_j^t) \quad (1)$$

$$\text{s.t. } \sum_{j \in F} \sum_{\substack{t \leq a_i + \tau_i - \delta'_{jk} \\ t \geq a_i + \delta_{kj}}} X_{ij}^t = Z_i \quad \forall i \in N \quad (2)$$

$$\sum_{i \in N} \mu_i X_{ij}^t d_i \leq G_j^t + B_j^t \quad \forall j \in F, \forall t \in T \quad (3)$$

$$G_j^t \leq \overline{G_j^t} \quad \forall j \in F, \forall t \in T \quad (4)$$

$$B_j^t \leq \overline{B_j^t} \quad \forall j \in F, \forall t \in T \quad (5)$$

La **funzione obiettivo** (1) cerca di massimizzare il guadagno del provider di rete, definito come ricavi meno costi. I ricavi del sistema provengono dai rewards dei task che il sistema accetta di eseguire (entro la deadline). I costi del sistema sono dovuti alla quantità di energia green e brown utilizzate per risolvere i task.

Il vincolo (2) è il vincolo di **accettazione** dei task. Se il sistema decide di accettare un task  $i$  ( $Z_i = 1$ ) allora la sommatoria delle porzioni del task assegnate alle varie facility in diversi timeslots risulterà 1. Analogamente se il task non viene accettato ( $Z_i = 0$ ) il sistema non deve assegnare porzioni del task alle facility. Come si può notare dalla sommatoria più interna, il task deve essere risolto tra  $a_i + \delta_{kj}$  e  $a_i + \tau_i - \delta'_{jk}$ , dove  $\delta_{kj}$  e  $\delta'_{jk}$  rappresentano rispettivamente il tempo necessario per inviare la porzione di task  $i$  dalla facility  $k$  alla facility  $j$  e il tempo per inviare il risultato del processamento dalla facility  $j$  alla facility  $k$ .

Il vincolo (3) è il vincolo di **capacità** sulle facility, che sostanzialmente impone che il carico di lavoro assegnato alle facility deve rispettare la capacità delle facility stesse. Come spiegato in precedenza, il sistema decide come suddividere la computazione di un task e la variabile decisionale che rappresenta questa suddivisione è  $X_{ij}^t$ . Ogni task è anche caratterizzato dalla dimensione del suo input  $d_i$  e dalla risorsa di CPU necessaria per elaborare un'unità di dati in input,  $\mu_i$ . La moltiplicazione di questi tre valori determina la quantità di input da elaborare, pesata secondo  $\mu_i$ . Il vincolo (3) ci dice quindi che, per ogni facility e per ogni timeslot, il carico di lavoro assegnatogli non deve superare la sue capacità energetiche in quel determinato momento. Si può fare una piccola precisazione sulle eventuali unità di misura utilizzate nelle ipotetiche implementazioni del vincolo. La dimensione di un task  $d_i$  può essere pensata in byte o come multipli del byte, mentre  $\mu_i$  è un numero intero e  $X_{ij}^t \in [0, 1]$ . In questo modo la parte sinistra del vincolo esprime una grandezza determinata direttamente dai task e quindi misurata in byte. Tuttavia la parte destra esprime una grandezza energetica, che rappresenta la capacità computazionale di un nodo. La diseguaglianza sembra

quindi mettere a confronto due valori di natura diversa, ma il modello è lasciato volutamente generale, in modo da poter generare diverse implementazioni dello stesso. Il concetto che sta alla base del vincolo è che il carico di lavoro determinato dai task non deve eccedere la capacità computazionale dei nodi. Anticipando il capitolo 3, l'implementazione da noi prodotta presenta KB sulla sinistra e MW sulla destra della diseguaglianza. Per confrontare propriamente le due grandezze è stata utilizzata una variabile di **energy consumption** ( $J/KB$ ), che esprime la quantità di joule necessari per processare un KB. La variabile moltiplica la sommatoria di sinistra, evidenziando la quantità di joule necessari per processare i task. Ricordando che un watt equivale ad un joule al secondo ed anticipando che nell'implementazione un timeslot equivale ad un minuto, basta dividere la parte sinistra per 60 per ottenere la stessa unità di misura da entrambe le parti.

$$\frac{\text{energy consumption}}{60} * \sum_{i \in N} \mu_i X_{ij}^t d_i \leq G_j^t + B_j^t \quad \forall j \in F, \forall t \in T \quad (6)$$

Infine, i vincoli (4) e (5) impongono che la quantità di energia green e brown utilizzate da una facility in un determinato timeslot non deve superare la rispettiva quantità disponibile.

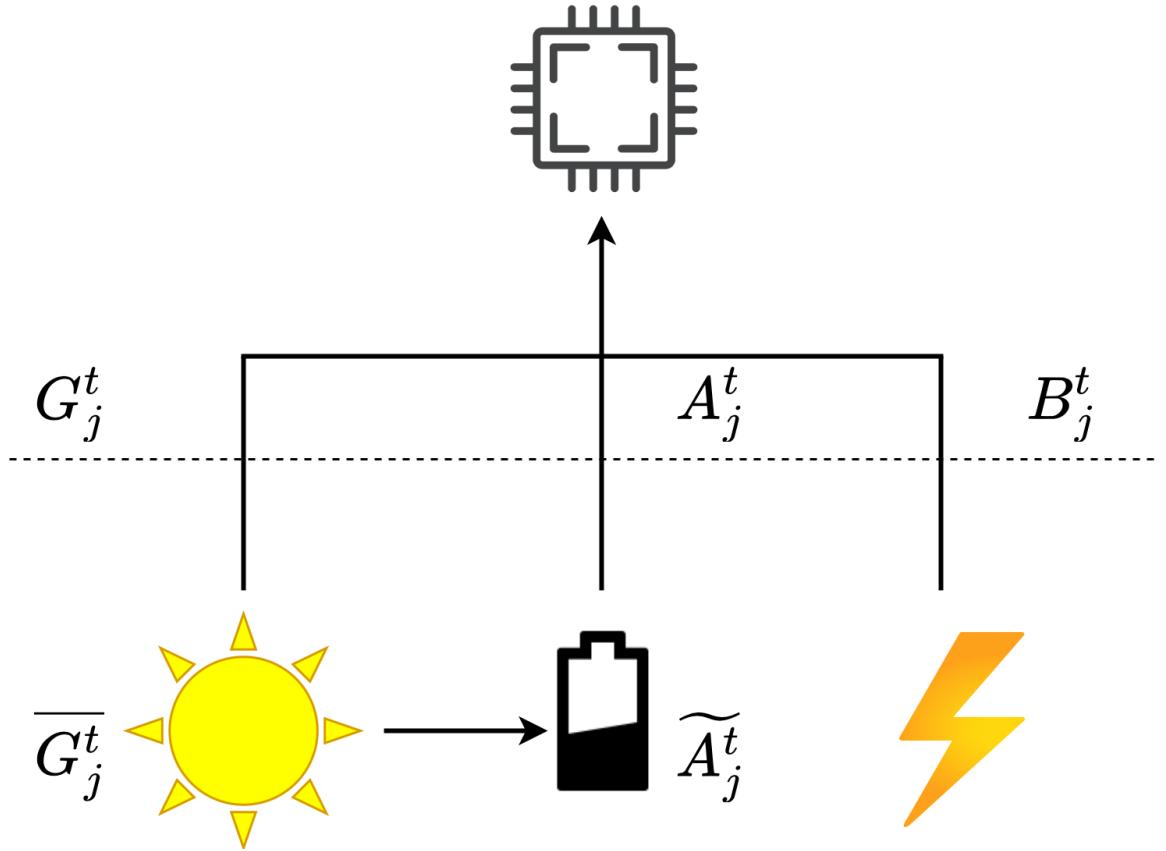
## 2.3 Modello green edge con accumulatori

Sulla base del sistema descritto nel capitolo 2.2 è stata ideata un'ulteriore modellazione del problema di task scheduling, che sostanzialmente è un'estensione del modello precedente alimentato da energia green e brown. Il nuovo modello prevede, oltre alle due fonti energetiche classiche, la presenza di accumulatori per ogni facility. In questo modo ogni nodo della rete ha a disposizione tre fonti di alimentazione energetica, al contrario delle due del modello precedente. Ogni facility può utilizzare direttamente l'energia green (chiamata harvesting in questo modello) oppure decidere di

immagazzinarla in un accumulatore per averla a disposizione nei timeslots successivi. Ovviamente ogni nodo continua a poter utilizzare l'energia brown. Il modello mantiene le variabili decisionali  $Z_i$ ,  $X_{ij}^t$  e  $B_j^t$  con il medesimo significato che hanno nel modello precedente (2.2). In aggiunta consideriamo le seguenti variabili decisionali:

- $H_j^t \in \mathbb{R}^+ \cup \{0\}$ : rappresenta il numero di risorse computazionali, alimentate direttamente da *energia green* (harvesting) prodotta al tempo  $t$  dalla facility  $j$ .
- $A_j^t \in \mathbb{R}^+ \cup \{0\}$ : rappresenta il numero di risorse computazionali, alimentate da *energia green* prelevata dalla batteria della facility  $j$  al tempo  $t$ .
- $\widetilde{A}_j^t \in \mathbb{R}^+ \cup \{0\}$ : rappresenta il livello di carica della batteria della facility  $j$  al tempo  $t$ .

In Figura 2 è rappresentato lo schema di alimentazione dei nodi in questa nuova versione del problema. Come nel modello precedente, la quantità di risorse computazionali totale disponibile in un nodo  $j$  è fissa nel tempo e dipendente dal nodo stesso. Cambiano invece le fonti di alimentazione energetiche. Ogni facility, al tempo  $t$ , dispone di  $H_j^t \leq \overline{H}_j^t$  risorse di elaborazione utilizzando direttamente l'energia green. A differenza del modello precedente, il nodo può adesso decidere di accumulare l'energia green disponibile, o parte di essa, su un sistema di accumulatori energetici. Le risorse di elaborazione disponibili al tempo  $t$  utilizzando l'energia accumulata sono  $A_j^t \leq \widetilde{A}_j^t$ , dove  $\widetilde{A}_j^t$  indica il livello di carica della batteria. Ogni batteria non può eccedere il suo livello massimo di carica pari a  $\overline{A}_j^t$ . Ogni facility  $j$  può ovviamente ricorrere all'utilizzo dell'energia brown, come nel modello precedente, avendo a disposizione al tempo  $t$  una quantità di risorse computazionali alimentate da energia brown pari a  $B_j^t$ . Nel modello precedente la quantità di risorse computazionali totale disponibile in un nodo  $j$  al tempo  $t$  era rappresentata dalla somma di  $\overline{G}_j^t$  e  $\overline{B}_j^t$ . Nell'estensione del modello con accumulatori la quantità di risorse computazionali totale disponibile in un nodo  $j$  al tempo  $t$  è specificata come la somma di  $\overline{H}_j^t$ ,  $\widetilde{A}_j^t$  e  $\overline{B}_j^t$ .



**Figura 2:** Schema di alimentazione con accumulatori

In questa modellazione, ogni nodo ha quindi tre fonti energetiche a disposizione, potendo utilizzare l'energia green appena prodotta, l'energia green accumulata nella batteria ed ovviamente l'energia brown.

Inoltre, come nel modello precedente, supponiamo che ogni fonte energetica sia caratterizzata dal suo prezzo unitario. L'energia green è suddivisa in *harvesting* e *accumulated*, con i rispettivi prezzi  $w_{hj}^t$  e  $w_{aj}^t$ , mentre l'energia brown costa  $w_{bj}^t$ . Assumiamo  $w_{hj}^t \leq w_{aj}^t \leq w_{bj}^t$ . Come in precedenza, assumiamo inoltre i costi di energia harvesting e accumulated fissi nel tempo, mentre il costo dell'energia brown varia in base alla domanda durante la giornata.

### 2.3.1 Modello

Di seguito il modello di PLI che rappresenta il problema con accumulatori:

$$\max \sum_{i \in N} r_i \cdot Z_i - \sum_{t \in T} \sum_{j \in F} (w_{hj}^t H_j^t + w_{aj}^t A_j^t + w_{bj}^t B_j^t) \quad (7)$$

$$\text{s.t. } \sum_{j \in F} \sum_{\substack{t \leq a_i + \tau_i - \delta'_{jk} \\ t \geq a_i + \delta_{kj}}} X_{ij}^t = Z_i \quad \forall i \in N \quad (8)$$

$$\sum_{i \in N} \mu_i X_{ij}^t d_i \leq H_j^t + A_j^t + B_j^t \quad \forall j \in F, \forall t \in T \quad (9)$$

$$H_j^t + A_j^t + B_j^t \leq \overline{C}_j \quad \forall j \in F, \forall t \in T \quad (10)$$

$$H_j^t \leq \overline{H}_j^t \quad \forall j \in F, \forall t \in T \quad (11)$$

$$A_j^t \leq \widetilde{A}_j^t \quad \forall j \in F, \forall t \in T \quad (12)$$

$$\widetilde{A}_j^t = \min\{\widetilde{A}_j^{t-1} - A_j^{t-1} + \overline{H}_j^{t-1} - H_j^{t-1}, \overline{A}_j\} \quad \forall j \in F, \forall t \in T \quad (13)$$

$$\widetilde{A}_j^0 = 0 \quad \forall j \in F \quad (14)$$

La **funzione obiettivo** (7), come nel modello precedente, cerca di massimizzare il guadagno dell'infrastruttura. La prima sommatoria rappresenta il ricavo totale derivante dai task accettati (e quindi processati) dal sistema. La doppia sommatoria di destra rappresenta invece la spesa che l'infrastruttura sostiene per processare i task. Come si può notare, i costi sono caratterizzati dalla quantità di energia harvesting, accumulated e brown utilizzate dal sistema.

Il vincolo (8) è il vincolo di **accettazione dei task**, identico al vincolo (2) del modello precedente. La somma delle porzioni del task  $i$  processate dalle facility nel tempo deve coincidere con  $Z_i$ , quindi uguale ad 1 se il task viene accettato e processato, 0 altrimenti.

Il vincolo (9) è il vincolo di **capacità**, tuttavia non è identico al corrispettivo vincolo (3) del modello precedente, anche se segue lo stesso principio. L'infrastruttura

attuale è alimentata tramite le tre fonti energetiche analizzate in precedenza, quindi il carico di lavoro delle facility nel tempo non deve eccedere la capacità allocata dal sistema nelle facility stessa, determinata da  $H_j^t$ ,  $A_j^t$  e  $B_j^t$ .

Il vincolo (10) impone che la somma delle quantità energetiche utilizzata da una facility  $j$  al tempo  $t$  non deve superare la capacità totale del nodo  $\overline{C}_j$ . Mentre nel modello precedente le fonti green e brown erano limitate dalla capacità energetica green e brown del nodo (vincoli (4) e (5)), nel modello attuale il nodo dispone di una capacità energetica totale  $\overline{C}_j$ .

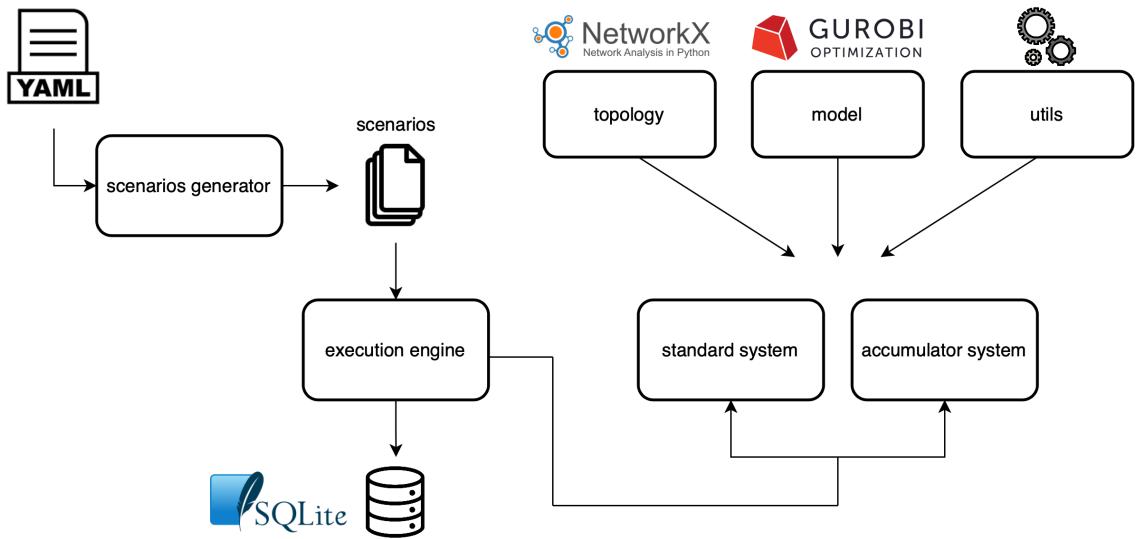
Il vincolo (11) limita l'utilizzo dell'energia harvesting rispetto alla capacità disponibile e il vincolo (12) limita la quantità di energia utilizzabile dalla batteria del nodo rispetto alla capacità corrente della batteria stessa.

Infine gli ultimi due vincoli riguardano il livello della batteria delle facility. In particolare il vincolo (13) specifica come avviene la ricarica della batteria nei nodi. La carica attuale in un nodo viene definita in funzione di alcuni valori e variabili del timeslot precedente, quali: il livello di carica della batteria, la quantità di energia della batteria utilizzata e la ricarica proveniente dall'energia green, nello specifico  $\widetilde{A}_j^{t-1} - A_j^{t-1} + \overline{H}_j^{t-1} - H_j^{t-1}$ . Essendo in ogni caso la quantità energetica accumulata nella batteria limitata da  $\overline{A}_j$ , il vincolo specifica il valore di ricarica come il minimo tra l'effettivo valore di un eventuale ricarica e la capacità massima della batteria. L'ultimo vincolo (14) impone semplicemente che il livello di carica per ogni batteria al tempo 0 sia nullo.

## Capitolo 3

# Implementazione

Il modello è stato implementato in *python* e in Figura 3 sono definiti i vari moduli principali che costituiscono il progetto. In particolare in *topology* è stata definita la rete delle facility, utilizzando la libreria **NetworkX**. In *model* troviamo invece tutte le funzioni necessarie per creare il modello di PLI, aggiungere le variabili decisionali, i vincoli e la funzione obiettivo. Il modulo si serve della libreria **Gurobi**, software specializzato che utilizza tecniche avanzate per risolvere problemi di programmazione lineare intera in modo efficiente. Il modulo *utils* contiene funzioni per generare i task, i valori delle varie facility e funzioni di import/export. I moduli *standard system* e *accumulator system* si servono dei tre moduli precedenti e creano l'istanza specifica da eseguire. Questi moduli generano il modello di PLI e lo risolvono, per poi esportare i risultati che verranno analizzati a posteriori. Questi due moduli sono lanciati a loro volta da un sistema che precedentemente crea una coda di scenari di esecuzione. In particolare il modulo *scenarios generator* si occupa di leggere un file **YAML** di configurazione e generare una serie di scenari diversi. Una volta che questi ultimi sono stati generati, il modulo *execution engine* si occupa di memorizzare ogni parametro degli scenari in un database dedicato, per poi lanciare parallelamente l'esecuzione degli scenari.



**Figura 3:** Schema di implementazione

### 3.1 File di configurazione

L’implementazione del modello permette di configurare i parametri delle varie simulazioni tramite un apposito file di tipo YAML (YAML Ain’t Markup Language). YAML è uno standard di serializzazione dei dati che presenta come caratteristica distintiva la leggibilità a misura d’uomo. YAML è utile e conveniente per diverse ragioni: è facile da leggere e scrivere perché utilizza una sintassi semplice e intuitiva basata su indentazioni. YAML può rappresentare dati strutturati in modo molto flessibile, consentendo di rappresentare dati di diversi tipi e strutture. YAML è supportato da molte librerie e strumenti di programmazione in diversi linguaggi di programmazione ed è compatibile con molti altri formati di dati, come JSON e XML.

Nell’appendice A è presente un esempio di file di configurazione utilizzato per la creazione di numerosi scenari di test. I parametri vengono poi analizzati e combinati dal modulo *scenarios generator* per creare diverse istanze da testare.

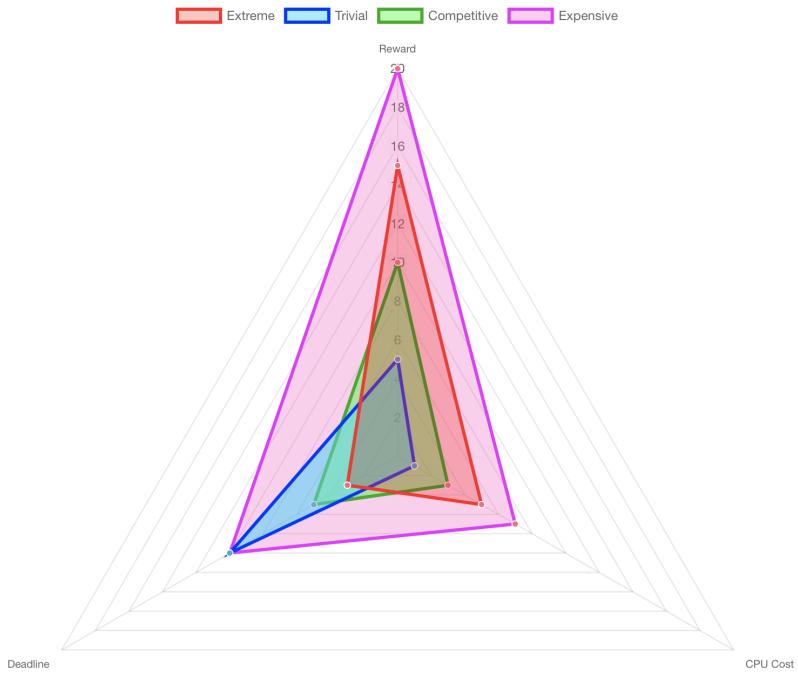
Il file di configurazione specifica la tipologia di modello che si vuole utilizzare, i timeslots ed il numero di task da generare nel test. Il file definisce anche la distribuzione con cui i task vengono creati, i nodi generatori dei task e la dimensione dell'output dei task. Infine vengono configurati alcuni valori che interessano la topologia dell'infrastruttura, come il consumo energetico unitario di processamento e i costi per le varie fonti di alimentazione.

I task sono inoltre classificati in diverse tipologie e i task di un determinato tipo vengono generati con una certa probabilità, che viene definita sempre nel file di configurazione. Questo comporta la generazione di diversi scenari, che differiscono anche per la composizione dei task che devono essere eseguiti dal sistema. Gli scenari possono quindi essere più o meno pesanti e difficili da processare a seconda della distribuzione di probabilità di generazione dei task.

La tipologia di un task definisce la sua dimensione di input, il reward, il costo computazionale e la sua deadline. In Figura 4 è possibile vedere le quattro tipologie definite, in particolare:

- *Competitive*: sono i task teoricamente più appetibili, con un basso costo computazionale, un buon reward ed una deadline nella media.
- *Trivial*: questi task hanno un basso reward, ma presentano una deadline ampia e un basso costo computazionale.
- *Expensive*: sono i task più difficili da eseguire a livello di costo computazionale, però presentano una deadline grande e sono molto redditizi.
- *Extreme*: sono i task più difficili da completare, in quanto la loro deadline è bassa, ma allo stesso tempo il costo computazionale è relativamente alto. Presentano un importante reward.

Le varie tipologie di task sono a loro volta configurate con altri file YAML.



**Figura 4:** Tipologie di task

## 3.2 Scenarios generator

Il modulo *scenarios generator* si occupa di caricare ad analizzare il file di configurazione e successivamente generare i diversi scenari con cui verrà testato il modello. Come si può notare nel file d'esempio 1, alcuni parametri sono espressi come semplici valori interi o float, mentre altri sono descritti con liste o con sotto-strutture. Il modulo *scenarios generator* si occupa di interpretare ogni parametro del file per generare correttamente gli scenari.

Prendendo come esempio il parametro `model_type`, che indica la tipologia di modello da testare, nel file 1 si può notare che il valore è espresso come lista [`'standard'`, `'accumulator'`]. Il modulo genera, a fronte degli stessi rimanenti parametri, due diversi scenari: uno in cui la tipologia del modello è `standard` ed uno in cui è `accumulator`. Questo vale per qualsiasi parametro specificato come lista nel file di

configurazione, come `number_of_task` e `input_nodes`. Il modulo esegue il prodotto cartesiano in modo da generare tutte le possibili combinazioni specificate nel file di configurazione.

Discorso diverso per il parametro `task_prob`, che descrive la probabilità di generare un task di una certa tipologia. In questo caso il modulo *scenarios generator* analizza le liste specificate per ogni tipologia combinandole per generare le probabilità dei task per ogni scenario. Leggendo il file di configurazione (Listing 1) dall'alto verso il basso, notiamo che, per il primo scenario, i task vengono generati secondo le seguenti probabilità:

- `trivial`: 40%
- `expensive`: 10%
- `competitive`: 30%
- `extreme`: 20%

Il parametro `arrival_time` specifica invece la distribuzione con cui i task vengono generati nel sistema. In questo caso la distribuzione è uniforme e il sistema genera i task dal primo all'ultimo timeslot, ma è possibile modificare la configurazione con un'altra distribuzione. Il modulo *scenarios generator* permette di specificare una serie di distribuzioni, tra cui uniforme, normale oppure costante.

I rimanenti parametri sono specifici dell'infrastruttura, come ad esempio i costi delle diverse fonti energetiche.

In Figura 5 è possibile vedere uno degli scenari generati dal modulo partendo dal file di configurazione.

```
{'arrival_time': 'uniform(0,timeslots)',  
 'battery_cost': 0.8,  
 'energy_consumption': 0.015,  
 'green_cost': 0.5,  
 'input_nodes': [0, 10],  
 'max_brown_cost': 20.0,  
 'min_brown_cost': 1,  
 'model_type': 'normal',  
 'number_of_tasks': 1000,  
 'output_size': 100,  
 'seed': 4046261434,  
 'task_names': ['trivial', 'expensive', 'competitive', 'extreme'],  
 'task_prob': [0.4, 0.1, 0.3, 0.2],  
 'timeslots': 96}
```

**Figura 5:** Esempio di scenario

### 3.3 Execution engine

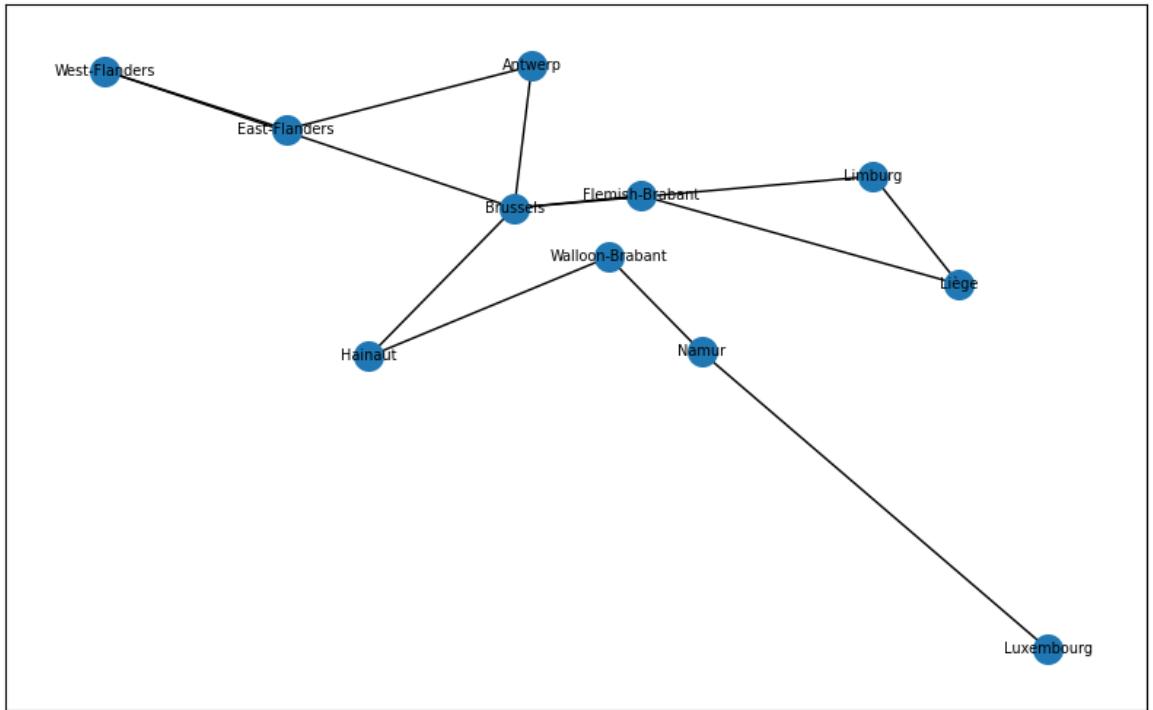
Il modulo *execution engine* si occupa di gestire una coda di esecuzione degli scenari e di aggiornare il database. In Figura 6 una porzione della tabella del database, contenente informazioni sugli scenari che sono stati eseguiti dal sistema. Ogni riga descrive uno scenario e vengono memorizzati tutti i parametri utilizzati per quello specifico scenario. In questo modo il sistema tiene traccia delle caratteristiche di ogni test eseguito, garantendone la riproducibilità. Il database contiene inoltre altri valori non derivanti dal file di configurazione, come l'`id` dello scenario oppure il `batch_timestamp`. Il modulo, dopo avere aggiornato il database, restituisce una coda di esecuzione di scenari, che il main andrà ad eseguire.

<b>id</b>	<b>direct...</b>	<b>batch...</b>	<b>co...</b>	<b>model...</b>	<b>timesi...</b>	<b>numbe...</b>	<b>arrival...</b>	<b>input...</b>	<b>output...</b>	<b>energy...</b>	<b>green...</b>	<b>batter...</b>	<b>min_br...</b>	<b>max_b...</b>	<b>task_n...</b>	<b>task_p...</b>	<b>seed</b>
1	c632ade9167...	c632ade9167...		08.03.2023 1...	1	normal	96	1000	uniform(0,..., [0, 10]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 4046261434
2	ea39f68762...	ea39f68762...		08.03.2023 1...	1	normal	96	1000	uniform(0,..., [0, 10]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 4054353473
3	2f810100d0...			08.03.2023 1...	1	normal	96	1000	uniform(0,..., [0, 10]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 1318308262
4	46577999ea...	46577999ea...		08.03.2023 1...	1	normal	96	1000	uniform(0,..., [0, 10]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 4062007777
5	a6c834e444...	a6c834e444...		08.03.2023 1...	1	normal	96	1000	uniform(0,..., [0, 10]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 1993379165
6	77239164e5...	77239164e5...		08.03.2023 1...	1	normal	96	1000	uniform(0,..., [0, 10]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 2083744314
7	2ed8dbbf2...	2ed8dbbf2...		08.03.2023 1...	1	normal	96	1000	uniform(0,..., [0, 10]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 2686922348
8	2606c91170...	2606c91170...		08.03.2023 1...	1	normal	96	1000	uniform(0,..., [0, 10]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 975210779
9	f665c10b35...	f665c10b35...		08.03.2023 1...	1	normal	96	1000	uniform(0,..., [0, 10]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 975210779]
10	d294a13b38...	d294a13b38...		08.03.2023 1...	1	normal	96	1000	uniform(0,..., [0, 10]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 937619434]
11	d40eae864...	d40eae864...		08.03.2023 1...	1	normal	96	1000	uniform(0,..., [1, 7, 9]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 4046261434]
12	dc05a1004...	dc05a1004...		08.03.2023 1...	1	normal	96	1000	uniform(0,..., [1, 7, 9]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 1318308262]
13	72a85f2156...	72a85f2156...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [1, 7, 9]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 1318308262]
14	12b41166ce...	12b41166ce...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [1, 7, 9]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 4062007777]
15	6ed1d109ef...	6ed1d109ef...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [1, 7, 9]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 1446909647]
16	05059581b...	05059581b...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [1, 7, 9]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 1993379165]
17	8e20230b19...	8e20230b19...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [1, 7, 9]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 2083744314]
18	2d0c389a1...	2d0c389a1...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [1, 7, 9]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 2686922348]
19	8593a1004...	8593a1004...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [1, 7, 9]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 937619434]
20	d6de0146aa...	d6de0146aa...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [1, 7, 9]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 937619434]
21	d474d82880...	d474d82880...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [6]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 4046261434]
22	ee0a44d46...	ee0a44d46...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [6]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 1318308262]
23	438f20a62...	438f20a62...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [6]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 4062007777]
24	5d9a92963...	5d9a92963...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [6]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 1318308262]
25	9259a1004...	9259a1004...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [6]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 937619434]
26	50415aae0b...	50415aae0b...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [6]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 1993379165]
27	e9c6e664c...	e9c6e664c...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [6]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 2083744314]
28	a7e66ca37...	a7e66ca37...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [6]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 2686922348]
29	8da51243...	8da51243...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [6]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 975210779]
30	b46d341d7...	b46d341d7...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [6]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 937619434]
31	31c0e4a04...	31c0e4a04...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [6]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 4046261434]
32	5993a1004...	5993a1004...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [6]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.4, 0.1, 0.3, ... 937619434]
33	337afe314d...	337afe314d...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [6]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.7, 0.1, 0.1, ... 1318308262]
34	1ae1cbdf69...	1ae1cbdf69...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [6]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.7, 0.1, 0.1, ... 4062007777]
35	0d2b5bb54...	0d2b5bb54...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [6]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.7, 0.1, 0.1, ... 1446909647]
36	33cd8b746...	33cd8b746...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [6]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.7, 0.1, 0.1, ... 1993379165]
37	f7a38e620b...	f7a38e620b...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [6]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.7, 0.1, 0.1, ... 2083744314]
38	33cd8b746...	33cd8b746...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [6]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.7, 0.1, 0.1, ... 2686922348]
39	4c89d9e03...	4c89d9e03...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [6]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.7, 0.1, 0.1, ... 975210779]
40	584104b33c...	584104b33c...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [6]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.7, 0.1, 0.1, ... 937619434]
41	b84540401c...	b84540401c...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [6]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.7, 0.1, 0.1, ... 4046261434]
42	6c5914212c...	6c5914212c...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [6]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.7, 0.1, 0.1, ... 4054353473]
43	9fb682836c...	9fb682836c...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [6]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.7, 0.1, 0.1, ... 1318308262]
44	271637778a...	271637778a...		08.03.2023 1...	0	normal	96	1000	uniform(0,..., [6]	100	0.015	0.5	0.8	1	20.0	[trivial], exp...	[0.7, 0.1, 0.1, ... 4062007777]

Figura 6: Database

### 3.4 Dati

In questa sezione viene descritto il processo di acquisizione dei dati, sulla base dei quali è stata definita la topologia di rete utilizzata per l'esecuzione degli scenari. L'acquisizione dei dati è stata cruciale in questo lavoro di tesi per definire le caratteristiche energetiche e la topologia di rete su cui abbiamo costruito il modello di task scheduling in ambiente green edge computing. Grazie ai dati acquisiti abbiamo potuto compiere una scelta accurata delle caratteristiche del modello e abbiamo potuto dimostrare l'efficacia del nostro approccio rispetto ad altre soluzioni. Inoltre, l'acquisizione dei dati ha consentito la valutazione della fattibilità del nostro approccio in condizioni reali. Per prima cosa si è cercato di riprodurre una topologia reale sui cui calare il modello di PLI. Sulla base dei dati raccolti è stata definita la seguente topologia in Figura 7 con 11 nodi che rappresenta il territorio del Belgio.



**Figura 7:** Topologia della rete

### 3.4.1 Dati sulla produzione energetica

Come dataset di riferimento per i livelli di energia green e brown è stata utilizzata la piattaforma *Open Data Elia* che fornisce un accesso semplice e aperto a tutti i dati della rete pubblica, compresa la produzione di energia, il carico, il bilanciamento, la trasmissione e la congestione. Elia Group è un operatore belga del sistema di trasmissione di elettricità ad alta tensione, che opera in Belgio e in Germania. In particolare sono stati utilizzati i dataset *Photovoltaic power production estimation and forecast on Belgian grid (Historical)* [9] e *Wind power production estimation and forecast on Belgian grid (Historical)* [10], che riportano rispettivamente informazioni sulla produzione energetica fotovoltaica e eolica nelle varie province del Belgio. In Figura 8 una porzione della tabella riguardante la produzione di energia solare.

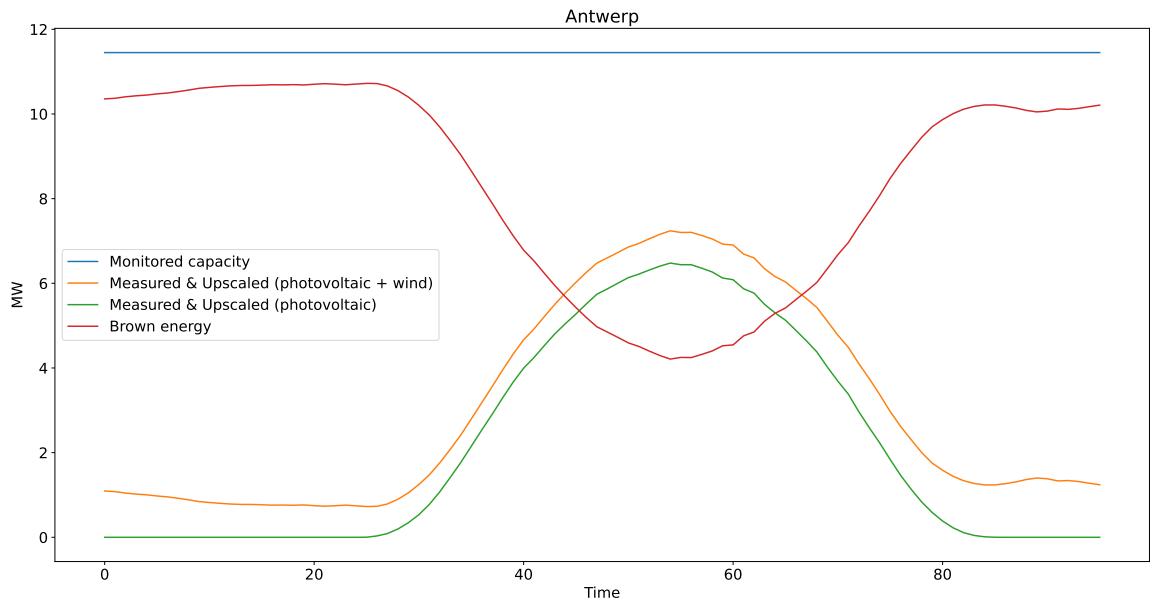
**Photovoltaic power production estimation and forecast on Belgian grid (Historical)**

Datetime	Resolution code	Region	Measured & Upscaled	Most recent forecast	Most recent P10	Mos
264	PT15M	Antwerp	68.09 MW	65.641 MW	48.455 MW	73.2€
265	PT15M	Hainaut	21.221 MW	22.381 MW	18.055 MW	27.9€
266	PT15M	Liège	35.243 MW	32.942 MW	28.238 MW	36.2€
267	PT15M	Belgium	474.523 MW	476.62 MW	365.408 MW	575.5€
268	PT15M	West-Flanders	62.966 MW	69.012 MW	49.995 MW	97.51
269	PT15M	Hainaut	28.094 MW	31.354 MW	24.094 MW	40.8
270	PT15M	Liège	51.359 MW	48.668 MW	39.991 MW	54.8
271	PT15M	Luxembourg	10.548 MW	12.492 MW	9.461 MW	15.2€
272	PT15M	Namur	15.39 MW	15.753 MW	11.79 MW	18.4€
273	PT15M	East-Flanders	89.379 MW	90.168 MW	74.105 MW	115.1€
274	PT15M	Walloon-Brabant	10.008 MW	9.801 MW	7.247 MW	11.53
275	PT15M	Brussels	10.959 MW	10.862 MW	9.888 MW	11.78
276	PT15M	Flemish-Brabant	43.962 MW	43.096 MW	36.928 MW	48.1€
277	PT15M	Flanders	348.165 MW	347.69 MW	262.937 MW	423.1
278	PT15M	Wallonia	115.399 MW	118.068 MW	92.583 MW	140.9
279	PT15M	Antwerp	100.09 MW	96.978 MW	64.237 MW	106.5
280	PT15M	Limburg	51.768 MW	48.436 MW	37.672 MW	55.4€

All dates and times are in Europe/Brussels time.

**Figura 8:** Photovoltaic power production

I valori interessanti per l’implementazione sono, in entrambe le tabelle, *Measured and Upscaled* e *Monitored capacity*, che rappresentano rispettivamente il valore energetico medio registrato nell’intervallo di tempo e la capacità produttiva totale. I dati utilizzati per l’implementazione sono quelli relativi al mese di agosto 2022, che riportano questi valori ad intervalli di 15 minuti. È stato condotto quindi un pre-processamento del dataset in cui uniamo i dati provenienti dalle tue tabelle per ottenere un unica fonte di alimentazione green. Successivamente, raggruppando i valori per provincia e per minuto del giorno ed eseguendo la media, è stata ricreata la giornata ideale del mese di agosto 2022, avendo quindi i valori per ogni provincia dalle 00:00 alle 23:59, ad intervalli di 15 minuti. Come si può vedere dall’esempio in Figura 9, abbiamo ottenuto i valori interessati in MW per tutta giornata, per un totale di 96 timeslot, ossia i quarti d’ora presenti in un giorno. Nell’implementazione del modello il valore *Measured and Upscaled* rappresenta quindi il livello di energia green registrato in un determinato timeslot, mentre *Monitored capacity* descrive la

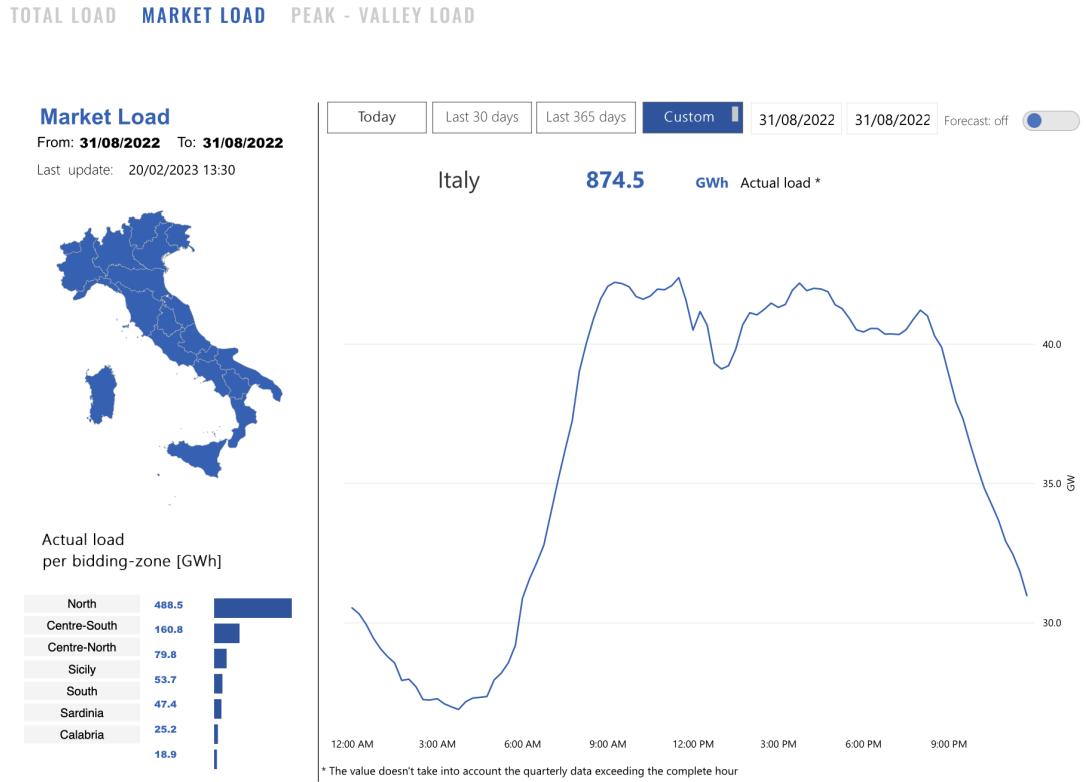


**Figura 9:** Potenza energetica giornaliera nella regione di Antwerp

capacità produttiva totale, che rimane costante durante tutta la giornata. Il livello di energia brown disponibile in una facility in un determinato timeslot viene calcolato come differenza tra *Monitored capacity* e *Measured and Upscaled*.

### 3.4.2 Dati sulla domanda energetica

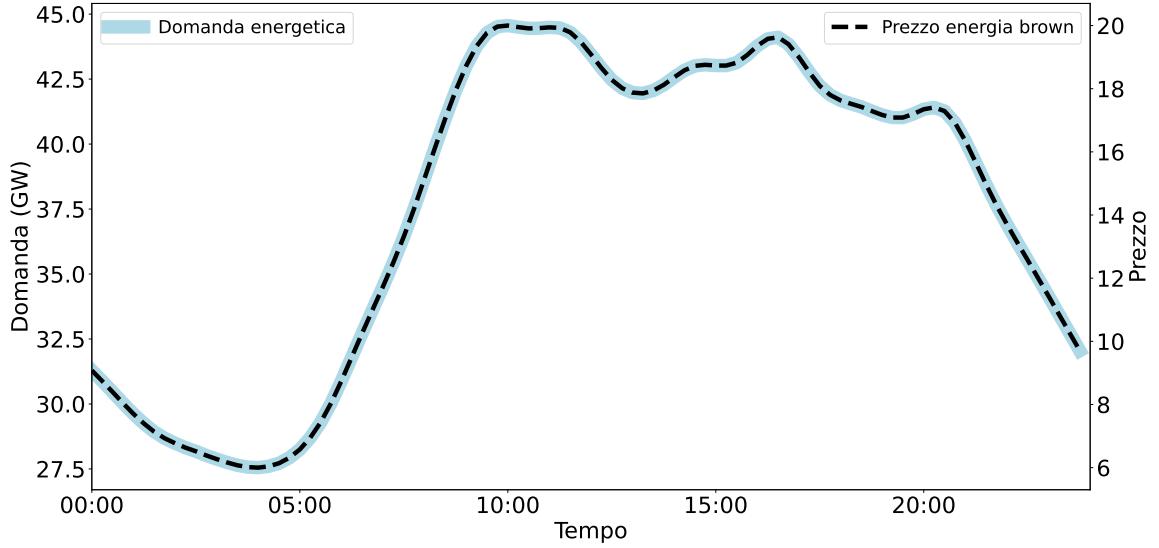
Come si può notare dalle funzioni obiettivo dei due modelli (1) e (7) le fonti energetiche sono associate ad un relativo prezzo di utilizzo e produzione. In particolare, nel modello con accumulatori, sono presenti tre diversi prezzi unitari energetici, rispettivamente per l'energia harvesting, accumulated e brown. A livello implementativo i prezzi dell'energia green e degli accumulatori sono costanti per tutte le facility e timeslot, pari a 0.5 e 0.8 rispettivamente. Il costo dell'energia brown è variabile e viene determinato in base alla domanda, aspettandoci un prezzo maggiore durante certe fasce orarie ed un prezzo minore in corrispondenza delle fasce in cui la domanda energetica è inferiore, come ad esempio durante la notte.



**Figura 10:** Market Load

Come riferimento per la domanda energetica giornaliera di un paese, sono stati utilizzati i dati forniti da *Terna* relativi al *Market Load* giornaliero italiano [11]. *Terna - Rete Elettrica Nazionale S.p.A.* è una società italiana operatrice delle reti di trasmissione dell'energia elettrica. Come si può vedere in Figura 10 sono stati considerati i dati relativi alla domanda registrata in un giorno di agosto 2022 in Italia.

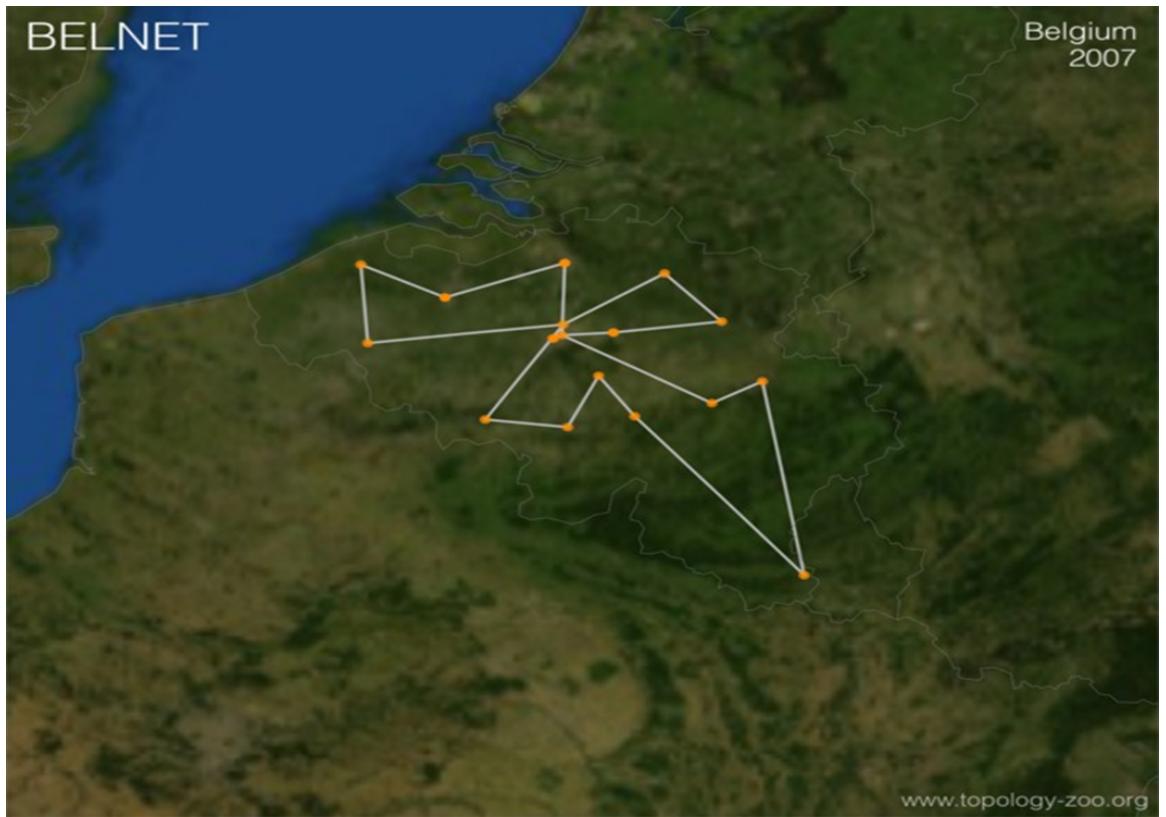
A questo punto, avendo definito un prezzo minimo e massimo per l'energia brown,  $w_{bj}^t$  viene calcolato in modo proporzionale rispetto alla domanda energetica. Nell'attuale implementazione il prezzo  $w_{bj}^t$  è lo stesso, a parità di timeslot, per ogni facility, con un prezzo minimo di 6 ed un prezzo massimo pari a 20. In Figura 11 è possibile notare come il prezzo e la domanda energetica seguano lo stesso andamento temporale, seppur rappresentando due concetti e valori differenti.



**Figura 11:** Rapporto tra domanda energetica e prezzo

### 3.5 Topology

In questa sezione viene esaminato il modulo *topology*, nel quale è definita la topologia della rete in Figura 7. Come spiegato nella sezione 3.4, la topologia attuale è stata creata basandosi sulla raccolta dei dati energetici delle province del Belgio. La realizzazione di questo modulo è stato uno dei primi step dell'implementazione del modello. I nodi della rete sono stati posizionati secondo le coordinate GPS delle province del Belgio, mentre gli archi del grafo sono pesati secondo le distanze in km tra le città. In questo modo l'implementazione consente di calcolare esattamente il tempo necessario per l'invio di una quantità di dati da una facility ad un'altra, come specificato nel primo vincolo del modello (2) con  $\delta_{kj}^t$  e  $\delta'_{jk}$ . Per definire i collegamenti tra i vari nodi del grafo, ci siamo ispirati alla rete *Belnet* in Figura 12. Belnet è la rete nazionale belga di ricerca ed istruzione che fornisce connettività e servizi associati per soddisfare le esigenze specifiche di istruzione superiore, ricerca e amministrazioni in Belgio.



**Figura 12:** Belgian National Research and Education Network (BELNET)

Il modulo è stato realizzato utilizzando la libreria **NetworkX**, un software open-source di Python utilizzato per l'analisi e la modellizzazione di reti complesse, come grafi, reti di flusso, reti stradali e molto altro ancora. La libreria fornisce strumenti per creare, manipolare e visualizzare una vasta gamma di tipi di reti. Fornisce inoltre numerosi algoritmi per l'analisi delle reti, tra cui algoritmi di centralità, di cammino minimo e di clustering. In sintesi, NetworkX è una libreria estremamente utile e flessibile per la manipolazione e l'analisi delle strutture di rete.

## 3.6 Model

Questo modulo contiene tutte le funzioni necessarie per definire il modello di PLI, utilizzando la libreria **Gurobi**. Il modulo, oltre a creare il modello, contiene un set di funzioni per aggiungere le variabili decisionali, i vincoli e la funzione obiettivo.

### 3.6.1 Gurobi

**Gurobi** è una libreria di ottimizzazione matematica che offre numerosi strumenti per la risoluzione di problemi di ottimizzazione lineare, quadratica, convessa e non lineare. **Gurobi** è una delle librerie di ottimizzazione più popolari e avanzate sul mercato, e viene ampiamente utilizzata in una vasta gamma di settori per la sua affidabilità e precisione nel risolvere problemi di ottimizzazione. **Gurobi** supporta diversi linguaggi di programmazione, tra cui C, C++, Java, Python e R. **Gurobi** offre un'alta velocità e precisione di calcolo ed utilizza diverse tecniche di risoluzione dei problemi di ottimizzazione, come *Branch & Bound* e *Branch & Cut*. La libreria inoltre offre la possibilità di utilizzare risorse di calcolo parallelo per ridurre i tempi di esecuzione. Grazie alla sua versatilità, affidabilità e capacità di risoluzione di problemi complessi, **Gurobi** è una soluzione ideale per chiunque abbia bisogno di risolvere problemi di ottimizzazione in modo efficiente e preciso.

### 3.6.2 Implementazione del modulo

Nell'appendice B è possibile esaminare con maggiore accuratezza l'implementazione del modulo *model*. In particolare viene illustrato come **Gurobi** è stato integrato al nostro progetto per creare il modello di PLI, aggiungere i set di variabili decisionali, i vincoli e la funzione obiettivo. Viene inoltre esaminato come l'implementazione del modulo fornisce funzioni diverse in base al modello di PLI che deve essere eseguito.

## 3.7 Utils

Il modulo *utils* è utilizzato da entrambe le implementazioni dei modelli e contiene tutte le funzioni di import/export ed inoltre permette di generare le facility e i task nel sistema. Nell'appendice C viene analizzata l'implementazione del modulo in dettaglio.

## 3.8 Standard System e Accumulator System

*Noram System* e *Accumulator System* sono i moduli che effettivamente eseguono il modello di PLI. I moduli prendono in input uno scenario generato precedentemente e settano il sistema utilizzando le funzioni di *topology*, *model* e *utils*. Una volta creata l'infrastruttura di rete ed il modello di PLI, il modulo esegue l'ottimizzazione del modello per infine esportare i risultati ed aggiornare il database.

## 3.9 Funzione di penalità

In fase di implementazione è stata realizzata una modifica del modello in modo da forzare il sistema alla saturazione, indipendentemente dalla potenziale perdita rispetto alla soluzione ottimale. All'interno del modulo *model* è stata inserita una variante della funzione obiettivo che penalizza fortemente il sistema nel caso in cui un task non venga accettato e processato. In questo modo il sistema è in un certo senso costretto ad accettare la maggioranza dei task fino ad arrivare alla saturazione. Questa nuova funzionalità ci permette di comprendere ancora meglio il corretto comportamento del modello, che punta in ogni caso a massimizzare il guadagno del provider di rete, ma che allo stesso tempo è operativo a livello computazionale ed energetico.

## Capitolo 4

### Euristica

In questo capitolo viene illustrata la prima versione dell'algoritmo risolutivo creato per risolvere il problema di task scheduling in ambito green edge computing. Le euristiche forniscono soluzioni approssimate che consentono di trovare una soluzione accettabile in tempi ragionevoli. L'utilizzo di euristiche risulta fondamentale in ambienti complessi, in cui la soluzione ottimale del problema è spesso troppo onerosa in termini di tempo e risorse. L'obiettivo del presente lavoro è sviluppare un'europeistica per il problema di task scheduling in ambiente green edge computing, avendo a disposizione l'upper bound fornito dal modello di PLI (2) per testarne l'ottimalità. Sono state prodotte due euristiche differenti per le rispettive tipologie di modello, che seguono però la medesima struttura e idea di base, ma utilizzano diverse fonti di alimentazione energetiche. La soluzione proposta rappresenta un contributo importante per il problema di task scheduling in ambiente green edge computing, garantendo una gestione ottimale delle risorse, una riduzione dei consumi energetici, una migliore qualità del servizio e un guadagno per il provider dell'infrastruttura. Inoltre, questa ricerca può fornire un punto di partenza per lo sviluppo di nuovi algoritmi e di soluzioni innovative per il task scheduling in ambienti complessi come quello del green edge computing.

## 4.1 Euristica Standard

Il primo algoritmo è quello utilizzato per affrontare il problema di task scheduling in un’infrastruttura green edge standard. L’architettura del sistema è la medesima del modello analizzato nel Capitolo 2.2, composta da un set di  $F$  facility alimentate da energia green e brown. L’idea su cui si basa l’euristica è la modifica del modello presentato in 2.2.2, trasformandolo da un modello di programmazione lineare intera (PLI) ad un modello di programmazione lineare (PL) per singolo task.

La principale differenza tra PL e PLI è che quest’ultima, come suggerito dal nome, consente solo soluzioni intere per le variabili del problema, mentre PL può utilizzare qualsiasi valore reale. In particolare per risolvere un problema di programmazione lineare intera è necessario l’utilizzo di algoritmi più complessi, che richiedono maggior tempo di calcolo rispetto a quelli utilizzati per un problema di semplice programmazione lineare. Questo è dovuto al fatto che la ricerca dello spazio delle soluzioni intere richiede più tempo rispetto alla ricerca locale utilizzata dalla programmazione lineare.

L’euristica proposta definisce un modello di PL per singolo task all’interno del sistema green edge. Il task viene gestito dall’algoritmo che lo schedula all’interno dell’infrastruttura, cercando di ottimizzare il consumo energetico e l’utile del sistema. Il guadagno del gestore di rete dipende, come sempre, dal riuscito processamento del task. Il procedimento è eseguito per ogni task, nell’ordine con cui si presentano al sistema. In particolare l’algoritmo si occupa di minimizzare il costo di processamento del task, che viene accettato nel caso in cui genera un guadagno economico per il gestore di rete. In caso negativo, il task viene scartato.

### 4.1.1 Modello

Consideriamo tre set di variabili di decisione continue:

- $X_j^t \in [0, 1]$ : rappresenta la porzione del task assegnato alla facility  $j$  al tempo  $t$ .
- $G_j^t \in \mathbb{R}^+ \cup \{0\}$ : rappresenta le risorse computazionali, alimentate da *energia green*, allocate nella facility  $j$  al tempo  $t$  per processare il task.
- $B_j^t \in \mathbb{R}^+ \cup \{0\}$ : rappresenta le risorse computazionali, alimentate da *energia brown*, allocate nella facility  $j$  al tempo  $t$  per processare il task.

Come in 2.2, assumiamo una rappresentazione discreta temporale e un insieme di  $F$  edge facility alimentate da energia green e brown. Inoltre tutte le rimanenti caratteristiche energetiche dell'infrastruttura restano le medesime. Di seguito il modello di PL per singolo task:

$$\max r - \sum_{j \in F} \sum_{t \geq a}^{t \leq a + \tau} (w_{gj}^t \cdot G_j^t + w_{bj}^t \cdot B_j^t) \quad (15)$$

$$\text{s.t. } \sum_{j \in F} \sum_{t \geq a + \delta_{kj}}^{t \leq a + \tau - \delta'_{jk}} X_j^t = 1 \quad (16)$$

$$\mu X_j^t d \leq G_j^t + B_j^t \quad \forall j \in F, \forall t \in \{a, a + d\} \quad (17)$$

$$G_j^t \leq \overline{G_j^t} \quad \forall j \in F, \forall t \in \{a, a + d\} \quad (18)$$

$$B_j^t \leq \overline{B_j^t} \quad \forall j \in F, \forall t \in \{a, a + d\} \quad (19)$$

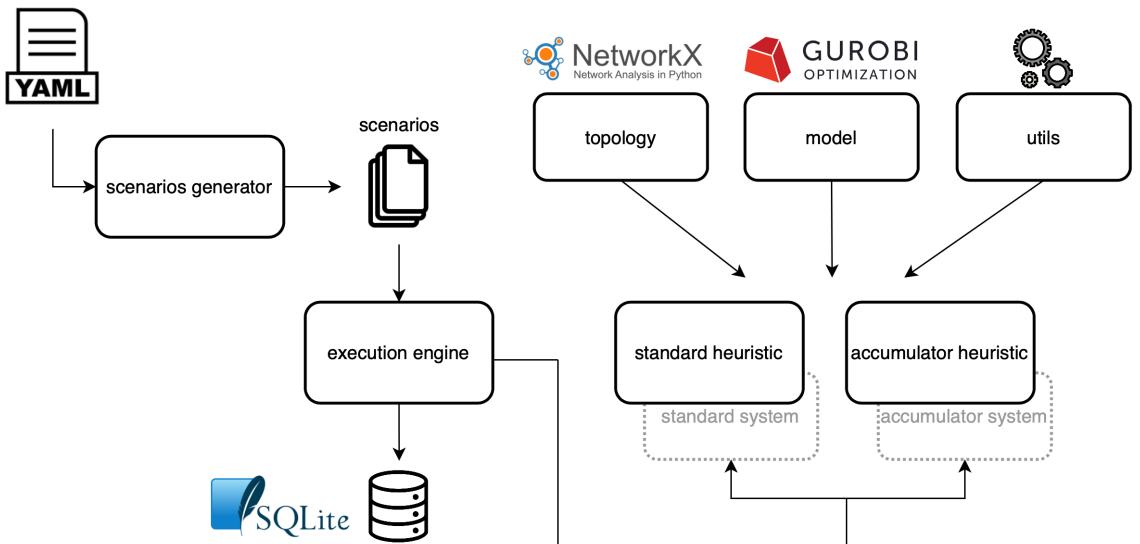
Come si può notare la struttura del modello è molto simile a quella proposta in 2.2, ma è l'assenza di variabili di decisione intere che rende il modello un modello di PL. Il modello in questione non presenta più una variabile di accettazione del task, in quanto è vincolato alla risoluzione del task in gestione. Il vincolo (16) impone

che la sommatoria delle porzioni del task che vengono processate dal sistema sia pari ad 1 e questo forza il processamento del task che il modello sta gestendo. Inoltre, gestendo un singolo task alla volta, è possibile vedere che il modello non presenta più alcun riferimento all'indice del task, cosa che nei modelli precedenti era rappresentata dall'indice  $i \in N$ . Notiamo ancora che il modello presenta un numero molto più basso di vincoli, essendo dedicato al singolo task. Il vincolo (16) è unico, mentre i set di vincoli (17), (18) e (19) dipendono dal numero di facility e dal tempo *di vita* del task. Il resto del modello è pressochè identico a quello presentato in 2.2, con la funzione obiettivo (15) che massimizza il guadagno del sistema e con il vincolo (17) che limita la capacità energetica utilizzabile nelle facility nel tempo. Infine i vincoli (18) e (19) limitano l'utilizzo di energia green e brown rispetto alle capacità massime del sistema.

Come precedentemente analizzato, l'algoritmo prevede la gestione sequenziale dei task nel loro ordine di arrivo nel sistema. Questo modello di PL viene quindi creato e risolto ad ogni iterazione dell'algoritmo e infine l'euristica restituisce la somma dei risultati prodotti dai modelli.

### 4.1.2 Implementazione

L'euristica è stata implementata realizzando un ulteriore modulo che è stato successivamente integrato all'implementazione complessiva del progetto. In Figura 13 vediamo come i due nuovi moduli, un per tipologia di modello, sono inseriti nel progetto e ricevono gli scenari di esecuzione dal modulo *execution engine*. I due nuovi moduli si occupano di leggere lo scenario che devono eseguire e di generare l'infrastruttura di rete e l'insieme dei task. A questo punto l'algoritmo vero e proprio inizia l'esecuzione, gestendo i task sequenzialmente nel loro ordine di arrivo. Grazie al modulo *model*, appositamente aggiornato per l'euristica, viene creato il modello di PLI, aggiungendo le variabili decisionali e i relativi vincoli.



**Figura 13:** Schema di implementazione euristica

Una volta risolto il problema di ottimizzazione, a fronte di una soluzione *feasible* e soprattutto vantaggiosa, l'algoritmo aggiorna la soluzione complessiva e avanza nella gestione del prossimo task. Tuttavia va specificato che, prima di avanzare nel processamento del task successivo, l'euristica deve aggiornare le capacità energetiche del sistema. In particolare nell'infrastruttura standard l'algoritmo deve modificare i valori delle strutture dati che rappresentano la capacità energetica green e brown nelle facility nel tempo. Al contrario dell'implementazione del modello standard di PLI (3.8), questa operazione di aggiornamento deve essere eseguita in quanto la risoluzione di uno scenario da parte dell'euristica prevede il processamento sequenziale di diversi modelli di PL tra loro indipendenti. Se per esempio l'algoritmo utilizza un quantitativo  $G_j^t$  di energia green in una facility  $j$  al tempo  $t$  per il processamento di un task, allora questo valore deve essere sottratto alla capacità totale green  $\bar{G}_j^t$  di quella facility in quel timeslot. In questo modo l'euristica, quando gestisce i task successivi, è aggiornata con lo scheduling prodotto nelle iterazioni precedenti, mantenendo la

coerenza sulle capacità energetiche del sistema nel tempo. Lo stesso discorso vale per l'energia brown. Nella versione standard dell'algoritmo è sufficiente aggiornare le strutture dati delle capacità energetiche  $\overline{G}_j^t$  e  $\overline{B}_j^t$  in base all'utilizzo effettivo  $G_j^t$  e  $B_j^t$ . Come vedremo nel capitolo 4.2 questo discorso diventa più complicato per la presenza degli accumulatori nel sistema.

## 4.2 Euristica con accumulatori

Il seguente algoritmo risolutivo affronta il problema di task scheduling in un'infrastruttura green edge con accumulatori. Così come in 4.1, l'euristica si basa sull'idea di trasformare il modello di PLI descritto nel capitolo 2.3 in un modello di programmazione lineare (PL) per singolo task. Anche in questo caso l'architettura del sistema è identica a quella descritta per il modello di PLI, composta da un insieme di  $F$  server, alimentati da energia harvesting, da accumulatori e da energia brown. Il task viene gestito da un algoritmo che lo schedula nell'infrastruttura, cercando di ottimizzare il consumo di energia e massimizzare il guadagno del gestore di rete. La procedura viene ripetuta per ogni task, in base all'ordine di arrivo nel sistema.

### 4.2.1 Modello

Il modello mantiene le variabili decisionali  $X_j^t$  e  $B_j^t$  con il medesimo significato che hanno nel modello precedente (4.1.1). In aggiunta consideriamo i seguenti set di variabili decisionali continue:

- $H_j^t \in \mathbb{R}^+ \cup \{0\}$ : rappresenta il numero di risorse computazionali, alimentate direttamente da *energia green* (harvesting) prodotta al tempo  $t$  dalla facility  $j$ .
- $A_j^t \in \mathbb{R}^+ \cup \{0\}$ : rappresenta il numero di risorse computazionali, alimentate da *energia green* prelevata dalla batteria della facility  $j$  al tempo  $t$ .

Anche in questo caso assumiamo una rappresentazione discreta del tempo e un insieme di  $F$  edge facility alimentate da energia green e brown e dotate di un sistema di accumulatori per immagazzinare l'energia green nel tempo. Tutte le rimanenti caratteristiche energetiche dell'infrastruttura restano le medesime dei modelli precedenti. Di seguito il modello di PL per singolo task:

$$\max \quad r - \sum_{j \in F} \sum_{t \geq a}^{t \leq a+\tau} (w_{hj}^t H_j^t + w_{aj}^t A_j^t + w_{bj}^t B_j^t) \quad (20)$$

$$\text{s.t. } \sum_{j \in F} \sum_{t \geq a+\delta_{kj}}^{t \leq a+\tau-\delta'_{jk}} X_j^t = 1 \quad (21)$$

$$\mu X_j^t d \leq H_j^t + A_j^t + B_j^t \quad \forall j \in F, \forall t \in \{a, a+d\} \quad (22)$$

$$H_j^t + A_j^t + B_j^t \leq \bar{C}_j^t \quad \forall j \in F, \forall t \in \{a, a+d\} \quad (23)$$

$$H_j^t \leq \bar{H}_j^t \quad \forall j \in F, \forall t \in \{a, a+d\} \quad (24)$$

$$A_j^t \leq \widetilde{A}_j^t \quad \forall j \in F, \forall t \in \{a, a+d\} \quad (25)$$

Le novità proposte dal modello rispetto al corrispettivo di PLI in 2.3.1 sono simili a quelle che il modello di PL in 4.1.1 propone rispetto a 2.2.2. Per prima cosa l'assenza di variabili di decisione intere rende il modello un modello di PL e non di PLI. Il vincolo (21) forza il processamento del task da parte del sistema, come nell'euristica standard. Inoltre anche in questo caso scompare il riferimento all'indice del task e diminuisce il numero di vincoli, che sono in funzione del *tempo di vita* del task in gestione. Il resto del modello segue la struttura del corrispettivo di PLI in 2.3.1: la funzione obiettivo massimizza il guadagno del gestore dell'infrastruttura e i vincoli (22) e (23) limitano la capacità energetica utilizzabile nelle facility nel tempo. Infine i vincoli (24) e (25) controllano la quantità utilizzata di energia harvesting e accumulated rispetto alle rispettive capacità del sistema. Va specificato che, al contrario del modello di PLI in 2.3.1,  $\widetilde{A}_j^t$  non è più espressa sotto forma di variabile decisionale, ma

è utilizzata dal modello a livello di input. Ricordiamo che  $\widetilde{A}_j^t$  rappresenta il livello di carica della batteria nelle facility nel tempo. Con la variabile decisionale scompaiono anche i relativi vincoli di ricarica degli accumulatori e del livello di carica di partenza. Questo cambiamento è dovuto dalla natura del nuovo modello, che essendo ora a singolo task, non necessita più di questa variabile ausiliaria che operava in funzione della quantità di energia harvesting accumulabile. Al contrario l'algoritmo può autonomamente tenere traccia dei livelli di carica delle batterie alla fine del processamento di ogni task.

L'algoritmo, come in precedenza, gestisce sequenziale i task nel loro ordine di arrivo nel sistema. L'euristica crea e risolve il modello di PL ad ogni iterazione e restituisce la somma dei risultati prodotti dai singoli modelli come risultato complessivo della risoluzione dello scenario.

#### 4.2.2 Implementazione

Per l'implementazione dell'euristica con accumulatori è stato realizzato un'ulteriore modulo che è stato integrato al progetto complessivo, come è possibile vedere dalla Figura 13. Così come in 4.1.2, anche questo modulo riceve lo scenario di esecuzione e genera successivamente l'infrastruttura di rete e i task. La struttura di base dell'algoritmo è pressoché la stessa dell'euristica standard: i task vengono gestiti sequenzialmente nel loro ordine di arrivo e per ognuno di questi viene creato un modello di PL coerente con 4.2.1. Il modello viene risolto e, a fronte di una soluzione *feasible* e vantaggiosa, viene aggiornata la soluzione complessiva dell'algoritmo.

Come in precedenza l'euristica risolve un problema di PL per ogni task e, per mantenere una coerenza energetica, prima di avanzare con il processamento del task successivo si occupa di aggiornare le capacità del sistema. In particolare, alla fine

di ogni iterazione, l'algoritmo aggiorna le strutture dati che rappresentano la capienza totale, la capacità green e i livelli delle batterie delle facility nel tempo. Inoltre, quando l'algoritmo gestisce un task con un *nuovo timeslot*, vengono aggiornati nuovamente i livelli energetici degli accumulatori, in quanto eventuale energia harvesting del timeslot precedente può potenzialmente essere immagazzinata nelle batterie. Così facendo l'algoritmo si aggiorna correttamente in funzione dello scheduling prodotto nelle iterazioni precedenti.

## Capitolo 5

# Analisi dei risultati

In questo Capitolo si presentano i risultati raccolti, le tecniche di analisi utilizzate e le conclusioni a cui si è giunti alla fine dell’analisi. Come spiegato nei Capitoli 3 e 4, le implementazioni dei modelli di PLI e delle euristiche sono state testate su numerosi scenari diversi tra loro, cambiando vari parametri del sistema per esaminare il conseguente comportamento del modello e dell’euristica. Lo studio che abbiamo effettuato si concentra in primo luogo sull’analisi dei risultati che il modello di PLI ha prodotto. In particolare ci assicuriamo che, a fronte di scenari differenti, il modello si comporti in maniera efficiente e coerente. L’implementazione del modello, nelle sue varie tipologie, è stata testata su numerosi scenari che differiscono tra loro per una serie di parametri come il numero di task oppure il loro punto di generazione a livello di nodi della rete. Inoltre, ricordando che abbiamo implementato quattro tipologie di task (Figura 4), gli scenari si differenziano anche per la distribuzione di probabilità con cui i task vengono generati secondo queste categorie. L’analisi effettuata sul modello di PLI si concentra sulla valutazione di risultati come la percentuale di task accettati, il risultato della funzione obiettivo e la saturazione del sistema nel tempo. In secondo luogo è stata condotta un’indagine sulle prestazioni dell’euristica creata(4), valutando la *bontà* dell’algoritmo in termini di risultato rispetto all’ottimo fornito dal modello di PLI.

## 5.1 Ambiente di test e generazione degli scenari

In ottica di testare le varie tipologie del modello sono stati ricreati una serie di scenari che differiscono per alcuni parametri di configurazione. In particolare sono stati generati diverse tipologie di scenario in base al numero dei task generati nel sistema, alla posizione di generazione dei task a livello di nodi di rete e in infine in base alla distribuzione di probabilità con cui i task vengono assegnati ad una particolare categoria.

Nella configurazione attuale uno scenario può generare 500, 1000, 2000 o 5000 task durante tutta la durata della simulazione. Al momento la generazione avviene in maniera Uniforme nel tempo, ma questo parametro è modificabile dal file di configurazione (3.1). Come punto di generazione invece sono presenti diverse opzioni, permettendo agli scenari di dare origine ai task in un unico nodo della rete, in due o tre nodi oppure in tutti i punti dell’infrastruttura. Anche in questo caso la scelta del nodo generatore viene eseguita in maniera Uniforme. Infine viene gestita la distribuzione di probabilità con la quale vengono creati i task secondo una certa categoria. Ricordando la Sezione 3.1 i task sono classificati in diverse tipologie. La tipologia di un task caratterizza la sua dimensione di input, il reward, il costo computazionale e la sua deadline. In altre parole la categoria di un task determina la difficoltà e appetibilità che contraddistingue il task in ottica di un processamento da parte dell’infrastruttura. Riassumendo velocemente i *competitive* sono i più appetibili, i *trivial* i più semplici da risolvere, gli *expensive* i più pesanti a livello computazionale e gli *extreme* i più difficili da risolvere soprattutto in termini di deadline. Variando la probabilità di generazione dei task secondo queste categorie, sono stati creati scenari più o meno facili o difficili da gestire. In particolare in Tabella 1 possiamo analizzare le quattro tipologie di scenari prodotti secondo le differenti distribuzioni.

Scenario	Task			
	Trivial	Expensive	Competitive	Extreme
Uniform	0.25	0.25	0.25	0.25
Plain	0.4	0.1	0.3	0.2
Soft	0.7	0.1	0.1	0.1
Tricky	0.2	0.4	0.1	0.3

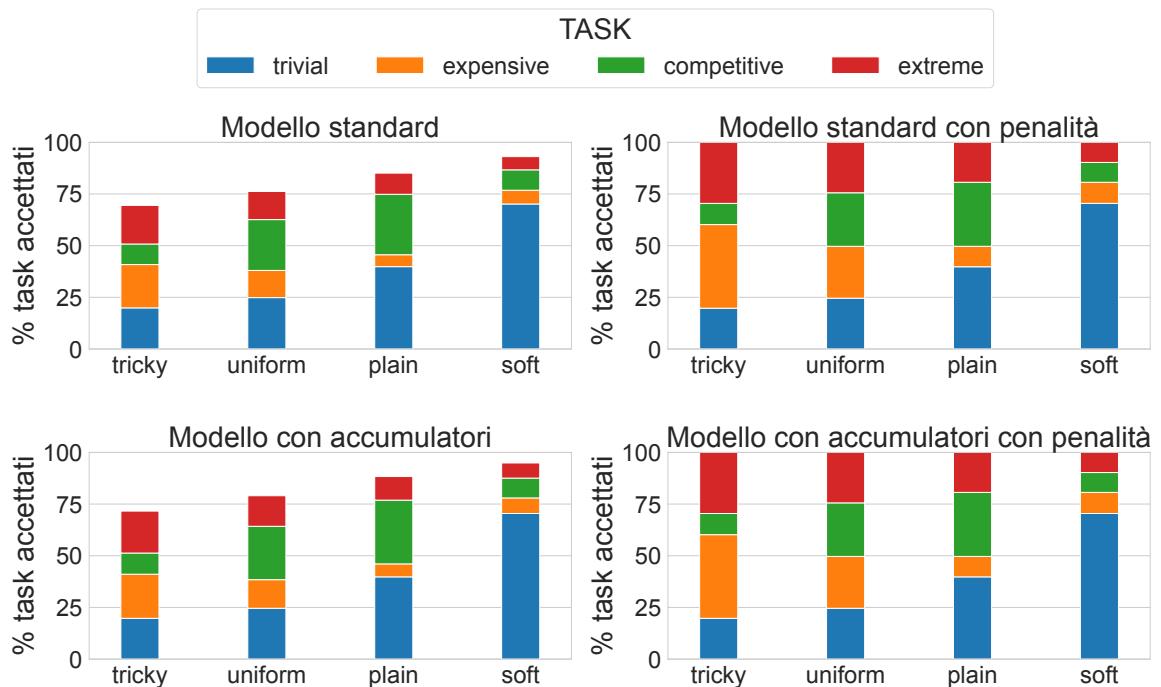
**Tabella 1:** Distribuzione di generazione dei task

La prima tipologia di scenario è **Uniform**, in cui ogni tipologia di task viene considerata con uguale probabilità. La seconda classificazione è **Plain**: qui più dei 2/3 dei task generati sono computazionalmente facili e controllabili dal sistema, essendo per il 40% e 30% rispettivamente di tipo *trivial* e *competitive*. Il restante 30% è contraddistinto da task più difficili e pesanti. La terza tipologia di scenario è quella **Soft**, nella quale il 70% è composto dai *trivial*, rendendo quindi lo scenario leggero a livello di carico computazionale. La restante parte è suddivisa Uniformemente per le altre tipologie di task. Infine notiamo lo scenario di tipo **Tricky**, ossia quello più impegnativo e faticoso per essere risolto. Questo scenario è caratterizzato da una prevalenza di task onerosi, ossia quelli di tipo *expensive* e *extreme* e solo il 30% è composto dai task *trivial* e *competitive*. Tuttavia questo scenario è potenzialmente quello più redditizio.

A fronte di tutte le possibili combinazioni di questi parametri (e dei rimanenti nel file di configurazione) sono stati generati una serie di scenari differenti su cui testare le varie tipologie del modello di PLI. Di seguito vengono analizzati i casi più significativi, in cui gli scenari sono composti da 1000 o 5000 task generati uniformemente in tutti i nodi del sistema.

## 5.2 Task accettati dal sistema

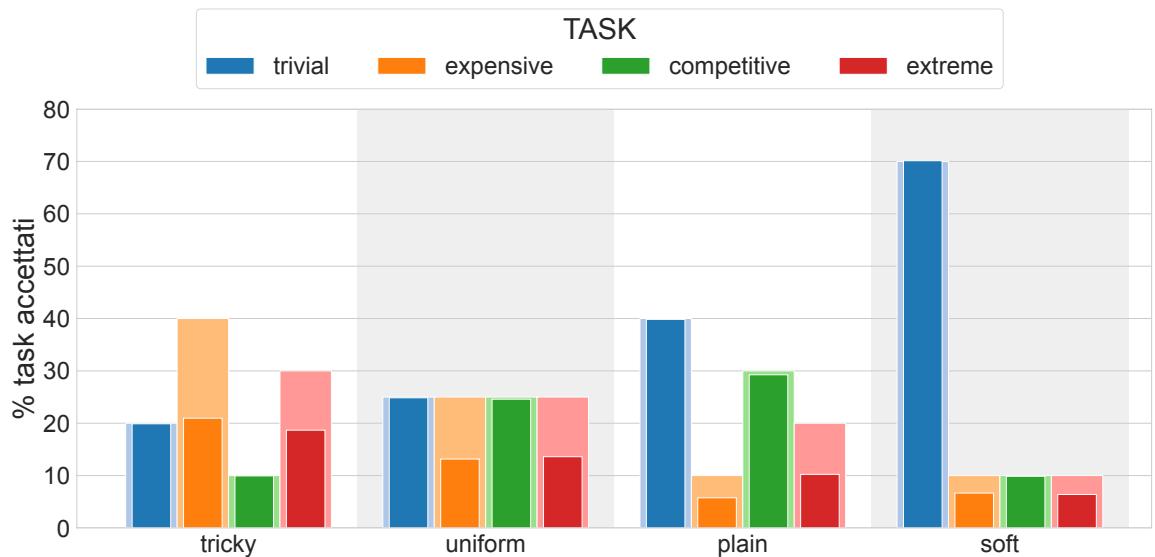
In questa sezione vengono proposti i risultati ottenuti analizzando il numero di task che il sistema accetta a fronte di diversi scenari. In Figura 14 possiamo esaminare la percentuale dei task accettati dai vari modelli di PLI. I task che sono stati generati in questo test sono 1000 e l'analisi viene condotta per ogni tipologia del modello, ossia il modello standard, il modello con accumulatori e le loro rispettive versioni con penalità. In particolare, ogni barra rappresenta la percentuale di task accettati secondo un determinato scenario, nell'ordine **Tricky**, **Uniform**, **Plain** e **Soft**. In questo grafico possiamo inoltre esaminare la suddivisione per tipologia di task sulla percentuale totale di task accettati, grazie al colore di ogni segmento della barra.



**Figura 14:** Percentuale di task accettati su un totale di 1000

Possiamo subito notare che il rate è tendenzialmente alto e solo in alcuni casi è di poco sotto all'80%. La media totale si aggira attorno al 91%, alzata sicuramente dai test eseguiti sui modelli con penalità, che registrano infatti una percentuale di accettazione più alta, con valori praticamente sempre uguali al 100%. Tralasciando i modelli con penalità e analizzando le diverse tipologie di generazione dei task possiamo vedere che lo scenario di tipo **Soft** permette al sistema, in tutte le sue tipologie, di accettare un numero elevato di task. Al contrario, lo scenario **Tricky**, essendo per definizione più pesante, produce un rate generalmente più basso. Notiamo infatti, considerando sempre i modelli senza penalità, che il tasso di accettazione è inversamente proporzionale alla difficoltà dello scenario. Infine è possibile notare che i modelli con accumulatori sono leggermente più efficienti nell'accettazione dei task rispetto ai corrispettivi modelli standard. Questo risultato è coerente con la modellazione del problema con accumulatori, in quanto il sistema può immagazzinare eventuale energia green ed utilizzarla successivamente ad un costo accessibile.

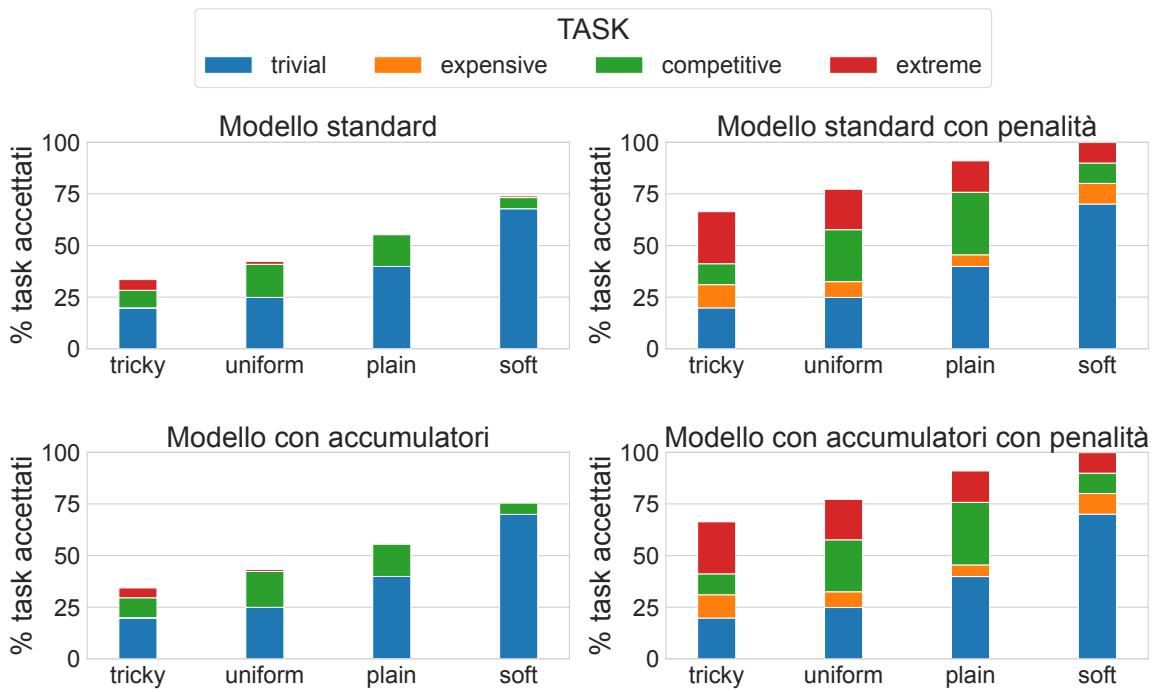
Analizzando la suddivisione dei task accettati, notiamo che i modelli con penalità hanno un rate di accettazione del 100% e quindi la divisione rispecchia sostanzialmente lo scenario che viene analizzato. I modelli senza penalità hanno suddivisioni pressoché simili tra di loro e i task che vengono maggiormente sacrificati sono quelli di tipo **expensive** e **extreme**. Approfondendo ulteriormente le analisi, si è proceduto a valutare la percentuale di accettazione in relazione alle varie tipologie di task, considerando esclusivamente i test condotti sul modello standard. I risultati ottenuti sono analizzabili in Figura 15, che offre una visione del comportamento del modello in funzione della tipologia di task. La figura presenta quattro raggruppamenti di barre in cui ogni raggruppamento raffigura l'analisi svolta su un determinato scenario. Ogni barra rappresenta una tipologia di task in cui la sezione di colore più acceso, che indica la percentuale di accettazione, ha come sfondo una parte di colore più spento, che rappresenta la quantità percentuale generata nel test per quel tipo di task.



**Figura 15:** Percentuale di accettazione per tipologia di task su un totale di 1000

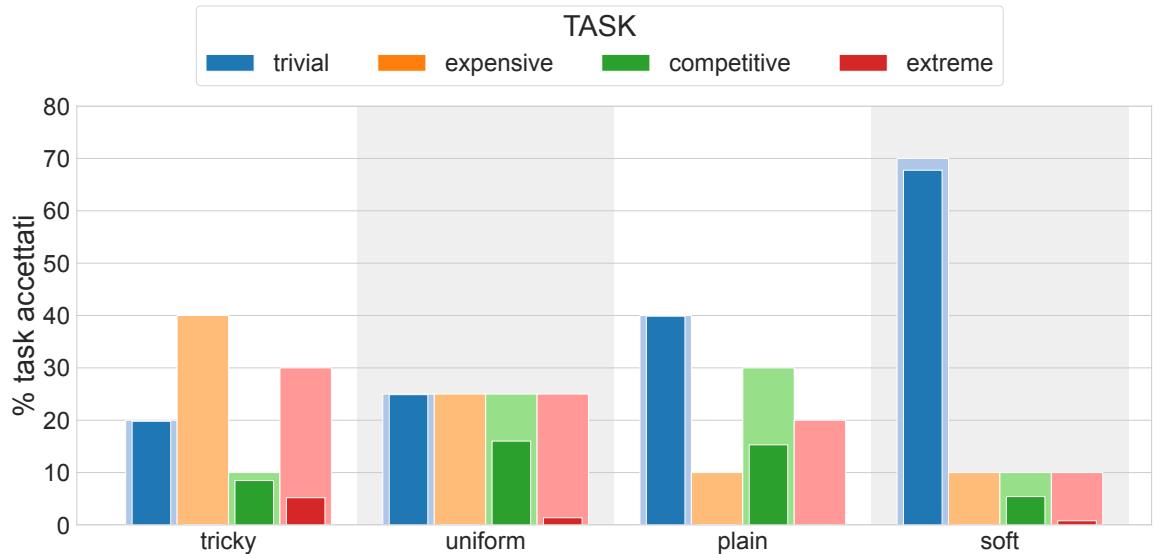
Tale rappresentazione grafica evidenzia quali sono i task maggiormente accettati e quelli che, al contrario, presentano maggiori difficoltà nell'essere elaborati. Se analizziamo lo scenario **Tricky** possiamo vedere che il 40% di task generati sono di tipo **expensive**, di cui ne sono stati accettati all'incirca la metà. Anche gli **extreme** vengono in parte rifiutati, mentre i **trivial** e i **competitive** vengono tutti processati. Guardando la figura nella sua totalità possiamo confermare quanto detto precedentemente: i task che vengono maggiormente sacrificati sono quelli più difficili da processare, mentre i task più appetibili vengono accettati nella loro maggioranza.

Analizziamo di seguito gli scenari che generano 5000 task considerando la Figura 16. I modelli senza penalità hanno rate percentuali minori rispetto al caso con 1000 task e la loro media è di circa 51%. Anche i modelli con penalità sono maggiormente in difficoltà e, tranne nel caso **Soft**, presentano valori percentuali tra il 60% e il 90%. Questa informazione ci indica che il sistema raggiunge la sua saturazione in questi casi, in cui lo scenario, oltre ad essere numeroso a livello di task, è pesante a livello computazionale.



**Figura 16:** Percentuale di task accettati su un totale di 5000

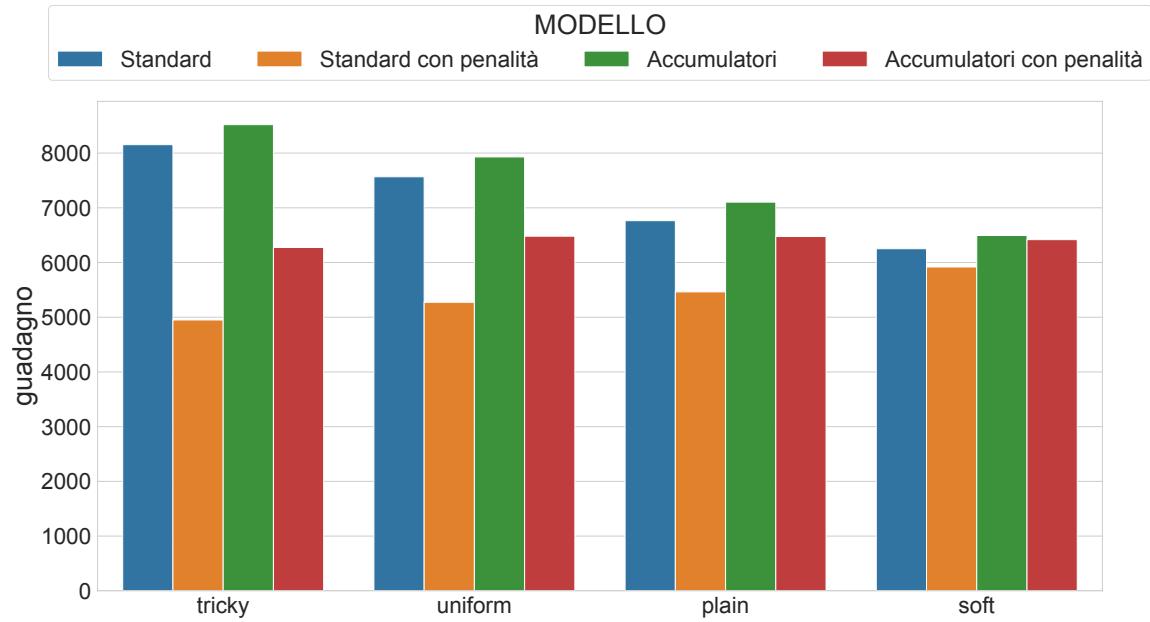
Esaminando la suddivisione dei task notiamo dalla Figura 16 che i task più pesanti vengono accettati con rate molto bassi, soprattutto nei modelli senza penalità. In quest'ultimi i task di tipo **expensive** non vengono praticamente mai accettati e gli **extreme** solo in parte quando lo scenario è **Tricky**. Anche in questo caso il modello è coerente e preferisce accettare i task computazionalmente meno pesanti. Possiamo vedere in Figura 17 la percentuale di accettazione per ogni tipologia di task per un modello standard. Il sistema tende ad accettare la maggioranza dei **trivial**, in quanto molto leggeri e facilmente allocabili. Anche i **competitive** hanno un buon rate di accettazione, in quanto seppur avendo costo computazionale leggermente maggiore, presentano un ottimo reward. I task meno appetibili risultano essere gli **expensive**, che non vengono mai processati. Gli **extreme** risultano poco invitanti, ma comunque minimamente accettati in determinati scenari.



**Figura 17:** Percentuale di accettazione per tipologia di task su un totale di 5000

### 5.3 Utile del sistema

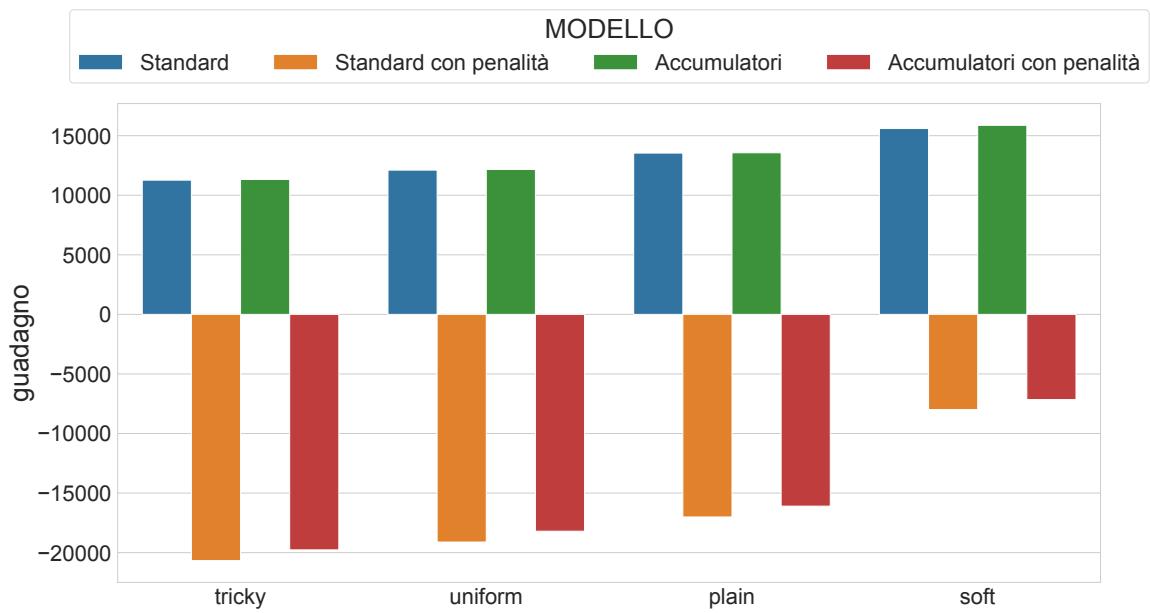
Questa sezione è dedicata all'analisi dei risultati dei modelli utilizzati per valutare il guadagno del gestore dell'infrastruttura in diversi scenari d'esecuzione. L'analisi viene condotta suddividendo i casi in cui il sistema deve processare 1000 e 5000 task, e la Figura 18 è dedicata alla presentazione dei risultati relativi ai 1000 task. È interessante notare che i modelli con penalità producono un utile minore rispetto ai corrispettivi modelli senza penalità. Questo conferma che i modelli con penalità, essendo forzati ad accettare il maggior numero di task possibile, generano un risultato non conveniente per il provider di rete. Infatti, accettando un numero minore di task, il provider produce un utile maggiore. Inoltre, i modelli con penalità presentano un alto dispendio energetico e processano i task anche quando questo non è opportuno per il sistema a livello di costi. Al contrario, i modelli senza penalità sono in grado di gestire i task in modo più efficiente, producendo un guadagno maggiore per il provider di rete.



**Figura 18:** Utile del sistema con 1000 task

Proseguendo con l’analisi delle distribuzioni di probabilità di generazione dei task, si può osservare un trend decrescente di guadagno che segue gli scenari nell’ordine **Tricky**, **Uniform**, **Plain** e **Soft**. Uno scenario difficile è comunque contraddistinto da una grande porzione di task redditizi e, seppur accettandone di meno in generale, il sistema è in grado di produrre un guadagno maggiore. Se il sistema non raggiunge la saturazione bisogna considerare che uno scenario pesante offre la possibilità di processare, oltre ad alcuni task leggeri e appetibili, task che sono oggettivamente prosperosi economicamente. Inoltre, è interessante notare che i modelli con accumulatori hanno un utile leggermente maggiore rispetto ai corrispettivi modelli standard e questo vale anche per le versioni del modello con penalità. La possibilità di accumulare energia green permette al sistema di processare successivi task ad un costo minore rispetto ad un sistema senza accumulatori. Riassumendo, il guadagno medio per i modelli senza penalità è di circa 7.300, ma l’analisi delle distribuzioni di probabilità di generazione dei task evidenzia che ci sono scenari in cui è possibile ottenere guadagni maggiori.

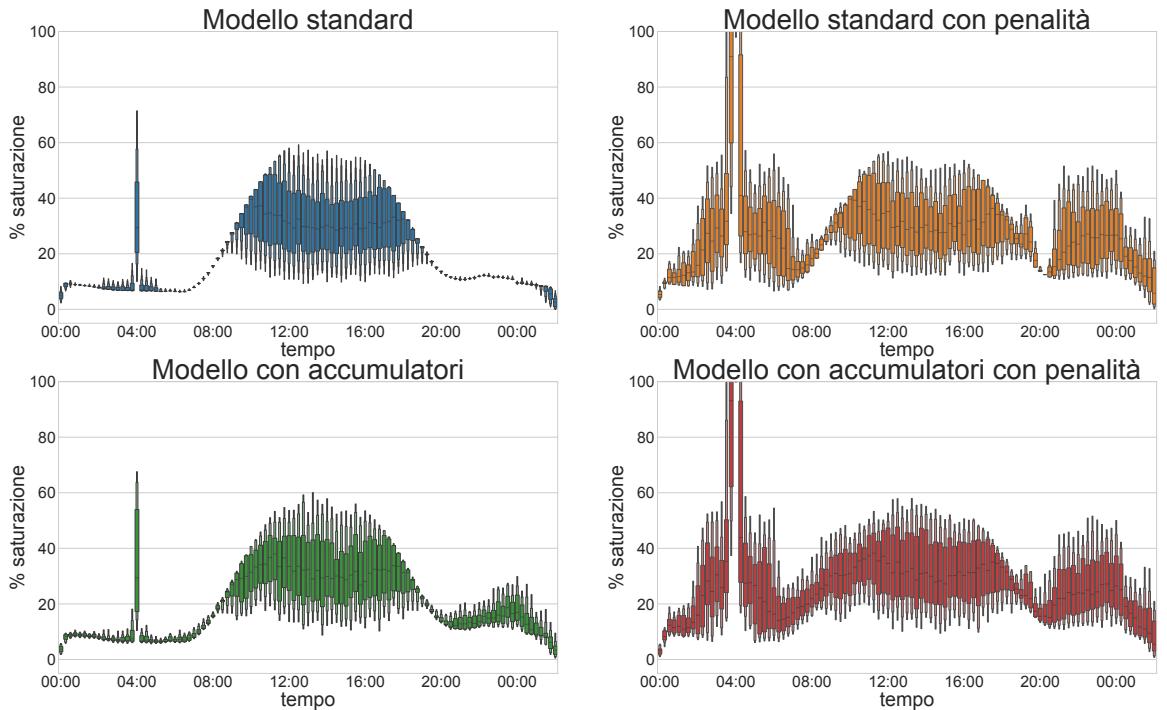
Consideriamo ora il comportamento dei modelli a fronte di scenari in cui vengono generati 5000 task e prendiamo come riferimento la Figura 19. Notiamo subito che in questa tipologia di scenario i modelli con accumulatori vanno sempre in perdita e cresce quindi il divario tra l'utile prodotto dai modelli con e senza penalità. Questo conferma che i modelli che sono forzati ad accettare la maggior parte dei task riducono il guadagno generabile dal processamento degli stessi, utilizzando fonti energetiche costose. In alcuni casi quindi il sistema non ha alcun vantaggio ad accettare un numero così elevato di task. Analizzando i modelli senza penalità vediamo che, avendo a disposizione molti più task, restituiscono un utile maggiore rispetto al caso precedente, con una media di circa 13.000. È interessante analizzare che, a livello di distribuzione di probabilità di generazione dei task, cambia il trend di guadagno, con un risultato maggiore rispetto a scenari più leggeri. Questa informazione suggerisce che il modello tende sempre a favoreggiare i task più leggeri e appetibili, per schedulare in secondo luogo, secondo le capacità energetiche residue, i task più pesanti.



**Figura 19:** Utile del sistema con 5000 task

## 5.4 Saturazione del sistema nel tempo

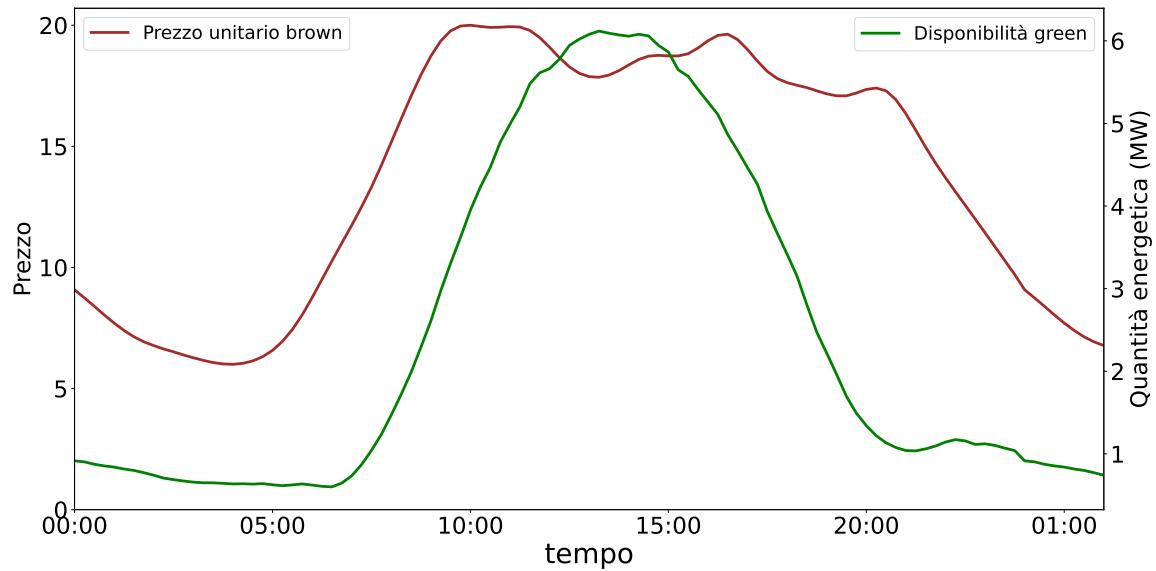
Di seguito viene condotta un'analisi sulla saturazione del sistema a fronte di diversi scenari d'esecuzione. Lo studio è stato realizzato considerando tutte le capacità energetiche delle facility ed esaminando in percentuale la saturazione del sistema nel corso del tempo. Come in precedenza, suddividiamo i casi in cui l'infrastruttura gestisce 1000 e 5000 task e analizziamo il comportamento dei diversi modelli. Consideriamo sempre il caso più significativo in cui i task vengono generati omogeneamente in tutti i nodi della rete. Per rappresentare la saturazione nel tempo viene prodotto un *boxplot* per ogni timeslot, che descrive la distribuzione di diversi test a fronte del medesimo scenario di configurazione. In Figura 20 possiamo analizzare il comportamento del modello negli scenari in cui il sistema genera 1000 task.



**Figura 20:** Saturazione del sistema nel tempo con 1000 task

In primo luogo è interessante esaminare l'andamento della curva nei modelli senza penalità. Dopo un unico picco iniziale, la curva segue un andamento a campana durante la fase centrale della giornata, per poi tornare a livelli bassi di saturazione alla fine del giorno. Questo comportamento del modello è correlato alla disponibilità energetica e all'andamento del prezzo per l'energia brown nel sistema. È analizzabile in tutte le figure un picco estremamente alto verso l'inizio della giornata, dovuto all'andamento del costo dell'energia brown. Durante la prima fase della giornata l'energia green è disponibile esclusivamente sotto forma di energia eolica, mentre l'energia solare è accessibile solamente più tardi (si veda Figura 9). Di conseguenza il sistema si satura moderatamente in queste fasce orarie, sfruttando la poca energia eolica disponibile. Allo stesso tempo decide di schedulare la maggior parte dei task in prossimità dei timeslot in cui il costo dell'energia brown è minimo (attorno alle 4:00 della mattina).

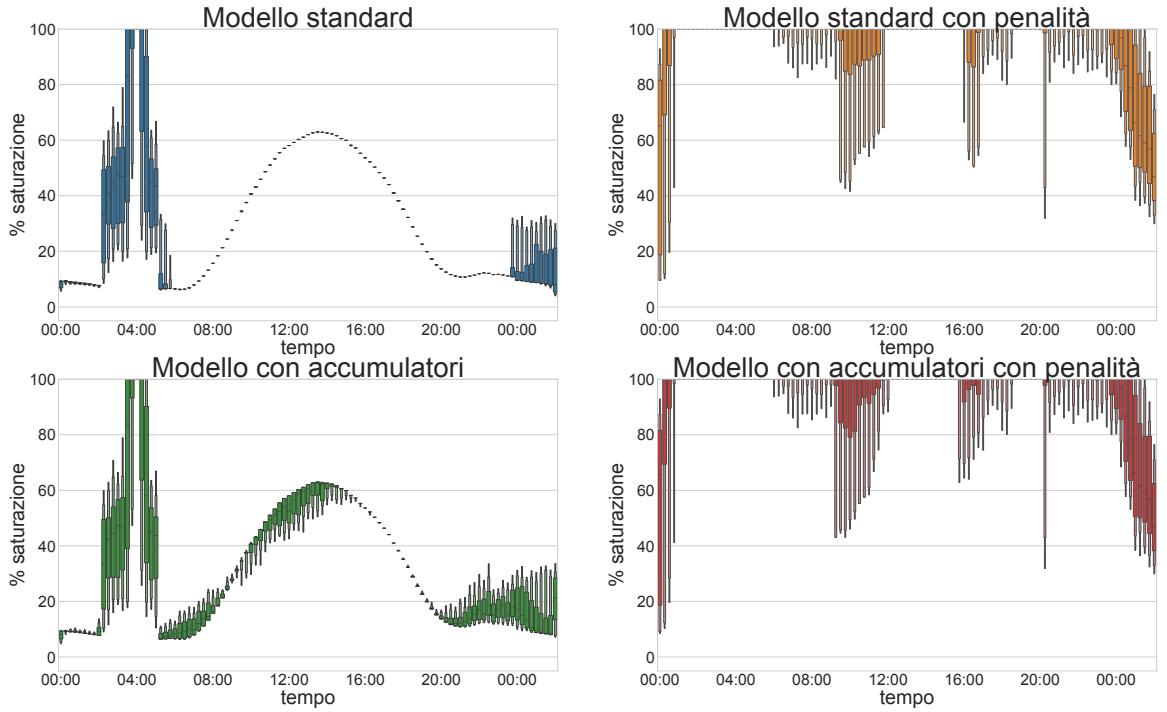
In Figura 21 è possibile analizzare l'andamento del prezzo unitario dell'energia brown e la disponibilità energetica green nella regione di *West-Flanders*.



**Figura 21:** Disponibilità green e prezzo dell'energia brown in West-Flanders

È inoltre possibile vedere in Figura 20 che la saturazione raggiunge valori più alti durante la fase centrale della giornata. Questa informazione conferma la coerenza del modello che, avendo a disposizione molta più energia green, si satura maggiormente, essendogli favorevole a livello economico. A questo punto la curva di saturazione scende nuovamente seguendo la diminuzione della disponibilità di energia solare. Anche in questo caso il comportamento di scheduling è coerente con il prezzo dell'energia brown, che decresce fino al termine della giornata. Sempre dalla Figura 20 si può vedere che i modelli con penalità sono complessivamente più saturi durante l'arco della giornata, in quanto forzati ad accettare il maggior numero di task possibile. Ricordando dalla Figura 14 che i modelli con penalità accettano sempre il 100% dei 1000 task, la Figura 20 ci mostra indicativamente quale sia la saturazione più efficiente per processare la totalità dei task. In altre parole, dai risultati dei modelli con penalità, possiamo capire quale sia il modo più efficiente e fruttuoso per forzare il processamento di tutti i 1000 task nel sistema.

In Figura 22 sono riportati i risultati dello studio della saturazione del sistema nel tempo per gli scenari che generano 5000 task. Analizzando i modelli senza penalità è possibile riconoscere, in maniera più accentuata, il trend di saturazione precedentemente esaminato con 1000 task. Ovviamente, gestendo un numero maggiore di task, si raggiungono percentuali di saturazione più alte a fronte degli stessi timeslot rispetto al grafico 20. Il primo picco di saturazione è molto più denso e distribuito su più timeslot. Il sistema infatti, dovendo processare molti più task, prova a concentrarli in prossimità del costo minimo dell'energia brown. I modelli senza penalità sono conformi al comportamento precedente, avendo sempre come obiettivo la massimizzazione del guadagno da parte del provider di rete.



**Figura 22:** Saturazione del sistema nel tempo con 5000 task

Al contrario analizzando i modelli con penalità notiamo all’istante alti livelli di saturazione durante tutto l’arco della giornata. In questi scenari l’infrastruttura si satura nella sua totalità, ma come analizzabile in figura, questo non accade sempre. Il motivo di questo risultato è riconducibile ad altre caratteristiche e vincoli del sistema, come la distanza dei nodi della rete e la deadline dei task.

Considerando tutte le valutazioni svolte, possiamo concludere che l’implementazione del modello si è dimostrata estremamente efficace e coerente con la sua formalizzazione originaria. Il modello si è rivelato altamente funzionale nel massimizzare il guadagno del gestore dell’infrastruttura, grazie alla capacità di schedulare i task nei momenti più opportuni in base alla disponibilità energetica e al prezzo di utilizzo. L’implementazione fornisce uno strumento efficace per calcolare e verificare la soluzione migliore di ottimizzazione del problema di task scheduling in ambiente green

edge computing. Grazie al modello realizzato è stato possibile ottimizzare al massimo l'utilizzo delle risorse a disposizione, massimizzando al contempo il guadagno del gestore del sistema e assicurando un'elevata qualità del servizio fornito e una gestione eco-sostenibile dell'infrastruttura.

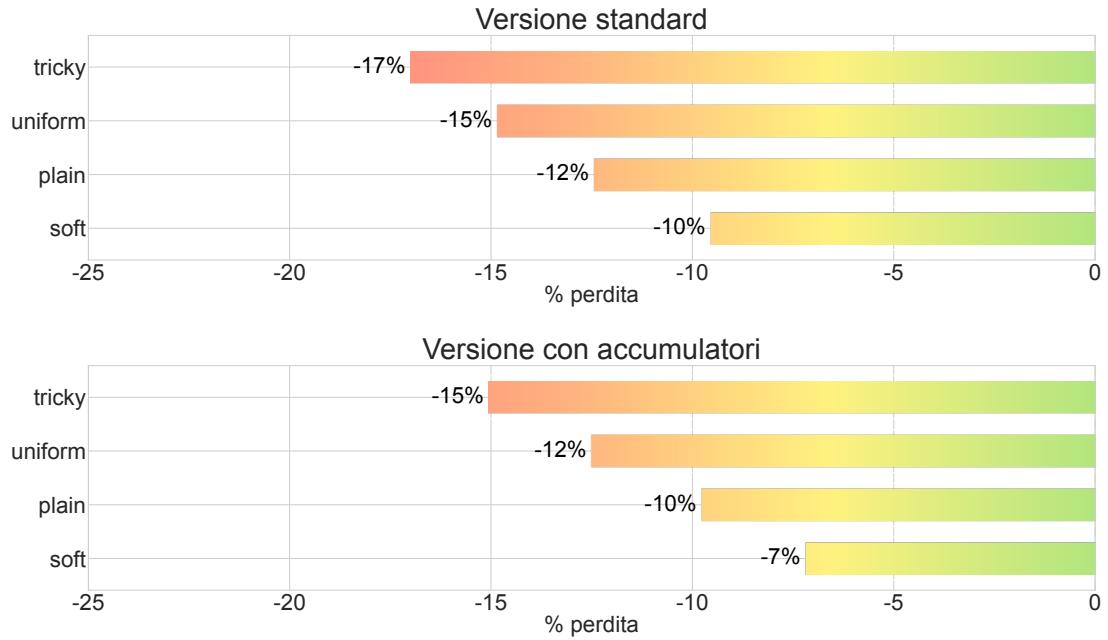
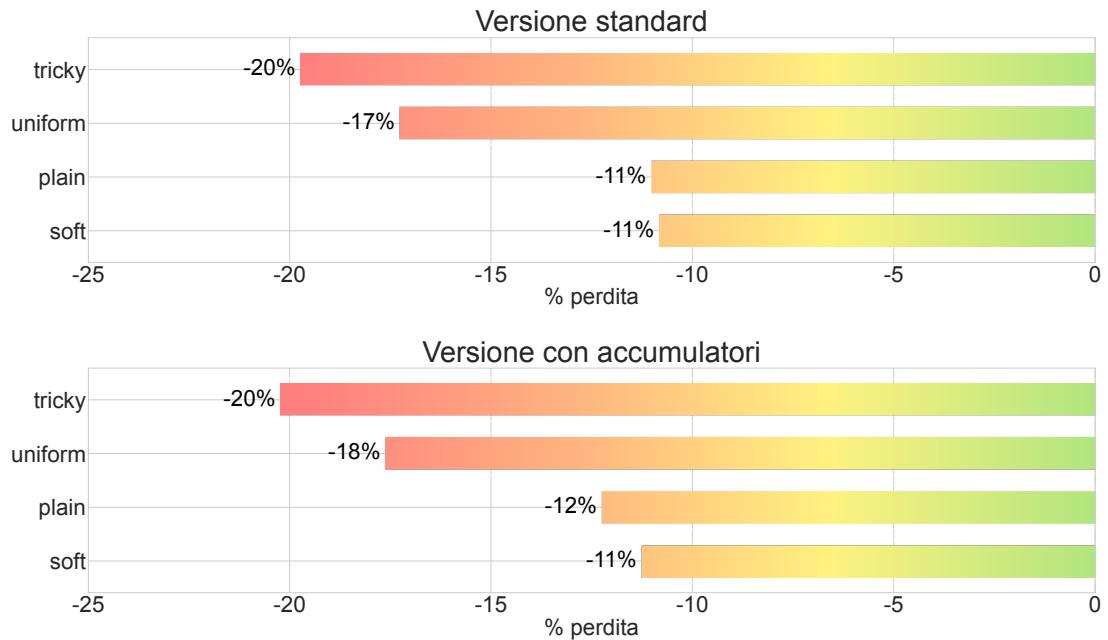
## 5.5 Euristica vs modello ottimo

In questa sezione vengono valutate le performance che l'euristica ideata produce. Viene valutata l'efficacia mettendo a confronto i risultati dell'euristica con i risultati che il modello di PLI (senza penalità) ha prodotto, calcolando la perdita percentuale di guadagno dell'algoritmo nei confronti della risoluzione ottima. Successivamente viene confrontato il tasso di accettazione dei task dell'euristica nei confronti del risultato ottimo di PLI.

### 5.5.1 Utile

In questa sezione viene analizzato l'utile prodotto dal nostro algoritmo risolutivo, confrontandolo con il corrispettivo ottimo fornito dal modello di PLI. L'analisi viene sempre condotta su scenari che producono 1000 e 5000 task, generati omogeneamente in tutti i nodi della rete.

In Figura 23 vediamo i primi risultati nel caso in cui gli scenari producono 1000 task. Questa analisi viene condotta per le versioni standard e con accumulatori, in quanto possiamo notare dal grafico che le perdite percentuali sono molto simili tra di loro nei vari scenari. L'euristica standard ottiene risultati leggermente peggiori di quelli prodotti dalla versione con accumulatori, ma in generale vediamo che le perdite percentuali sono sempre al di sotto del 17%. Si può notare una tendenza dell'euristica ad essere più efficace negli scenari **Soft**, mentre è più in difficoltà in quelli **Tricky** e **Uniform**.

**Figura 23:** Perdita percentuale dell'euristica con 1000 task**Figura 24:** Perdita percentuale dell'euristica con 5000 task

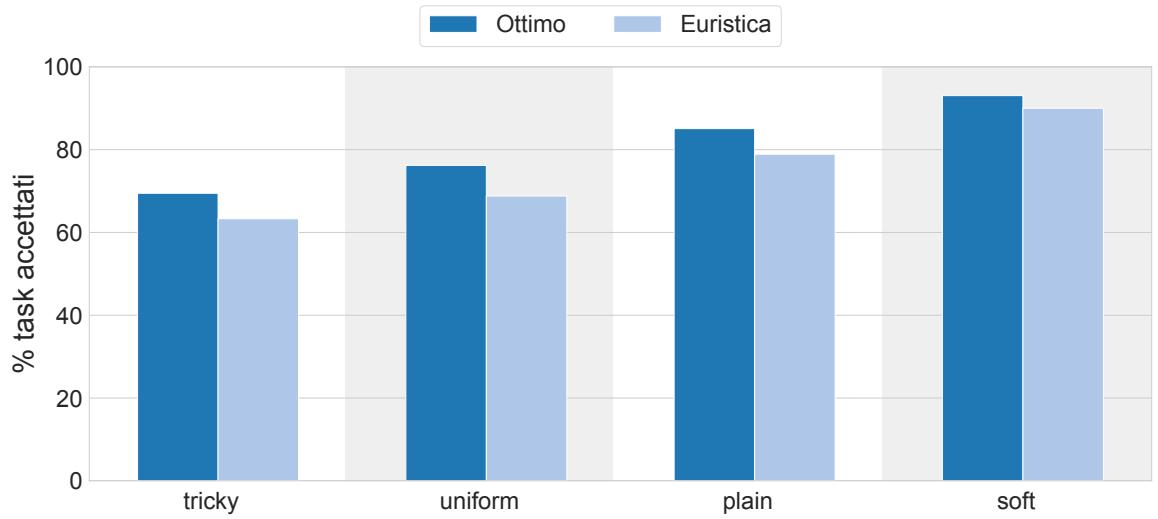
Questi risultati preliminari sono molto interessanti poiché ci permettono di fare alcune prime considerazioni sulle prestazioni dell’euristica in diverse situazioni. Le analisi finora condotte sono promettenti e suggeriscono che il nostro algoritmo per forma discretamente bene, con perdite percentuali sempre al di sotto del 17%, e in scenari più leggeri anche sotto al 10%.

In Figura 24 possiamo analizzare i risultati prodotti dall’euristica negli scenari che producono 5000 task. Anche in questo caso l’analisi viene condotta su entrambe le versioni dell’algoritmo, quella standard e quella con accumulatori. Come precedentemente visionato, anche in questo i risultati ottenuti dalle due versioni sono molto simili tra di loro. Notiamo immediatamente il peggioramento dell’algoritmo a fronte di uno scenario pesante e con un numero maggiore di task. Tuttavia il peggioramento rispetto agli scenari con 1000 task è lieve e la perdita sull’utile non supera il 20%. In particolare, come visto precedentemente in Figura 23, la perdita di guadagno è direttamente proporzionale alla difficoltà dello scenario di task da gestire.

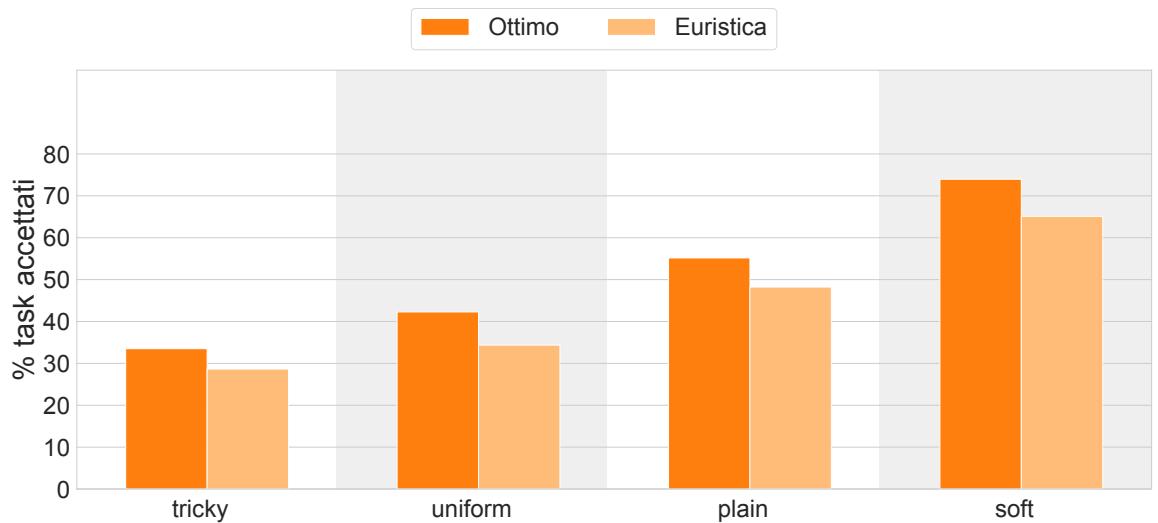
### 5.5.2 Task accettati

In questa sezione viene esaminato il tasso di accettazione dei task che l’euristica produce, confrontando sempre i risultati ottenuti con i rispettivi rate prodotti dalla risoluzione ottima del modello di PLI. Anche in questo caso, l’analisi viene condotta su scenari che producono 1000 e 5000 task, generati omogeneamente in tutti i nodi della rete.

Iniziando con l’analisi degli scenari che generano 1000 task, possiamo vedere i primi risultati in Figura 25. In ogni sezione del grafico, che si riferisce ad una tipologia di scenario diverso, vengono mostrate due barre, che indicano la percentuale di task accettati, dal modello di PLI e dall’euristica, sul totale dei task generati nel sistema. In tutti e quattro le tipologie di scenario, l’algoritmo si avvicina di molto dal risultato prodotto dalla soluzione ottima.



**Figura 25:** Tasso di accettazione task dell'euristica con 1000 task



**Figura 26:** Tasso di accettazione task dell'euristica con 5000 task

Questa informazione è incoraggiante in quanto ci suggerisce che il nostro algoritmo, oltre ad avere una perdita di guadagno discretamente bassa, si comporta similmente al modello di PLI a livello di accettazione dei task. Inoltre, come visto nei grafici precedenti, il valore di task accettati è inversamente proporzionale alla difficoltà dello scenario, sia per la risoluzione ottima che per l'algoritmo risolutivo.

In figura Figura 26 possiamo analizzare il tasso di accettazione dei task negli scenari che generano 5000 task. I risultati confermano che anche in questo caso l'euristica si comporta in modo simile al modello di PLI, accettando sempre un numero di task lievemente inferiore. Anche in questo grafico vediamo che il valore di task accettati è inversamente proporzionale alla difficoltà dello scenario, sia per la risoluzione ottima che per l'algoritmo risolutivo.

Le analisi condotte hanno permesso di valutare le prestazioni dell'algoritmo in termini di guadagno che il sistema percepisce e di numero di task che il sistema accetta. I risultati ottenuti indicano che l'algoritmo da noi ideato si comporta molto bene rispetto alla soluzione ottima del problema di task scheduling in ambiente green edge computing. In particolare, l'algoritmo riesce a raggiungere un guadagno molto vicino a quello ottenibile con la risoluzione ottima e riesce ad accettare un numero elevato di task. In conclusione, l'euristica si comporta molto bene rispetto alla risoluzione ottima del problema, dimostrando un'efficacia elevata sia in termini di guadagno che di accettazione di task. Questi risultati sono promettenti e rappresentano una solida base per lo sviluppo di soluzioni sempre più efficienti ed efficaci per la risoluzione del problema in esame.

# Conclusioni

In questo lavoro di tesi è stato realizzato un modello di ottimizzazione per il problema di task scheduling in ambiente green edge computing. Inizialmente è stato condotto uno studio preliminare dei lavori presenti in letteratura correlati al nostro progetto, che hanno ispirato e orientato la formalizzazione del problema. Successivamente è stato definito formalmente un modello di programmazione lineare intera per affrontare il problema di task scheduling in ambiente green edge computing. Il modello massimizza il guadagno del gestore dell'infrastruttura a fronte del processamento dei task che il sistema riceve, garantendo allo stesso tempo un consumo energetico ecologico-sostenibile, grazie alla capacità dei nodi della rete di essere alimentati anche da fonti di energia rinnovabili. Il modello è stato successivamente esteso permettendo alle facility dell'infrastruttura di immagazzinare l'energia green attraverso un sistema di accumulatori. Dopo aver raccolto una serie di dati attendibili su produzioni energetiche e variabilità del costo dell'energia, i modelli sono stati implementati in *python*. Per prima cosa si è ricreata la topologia della rete sulla base dei dati raccolti e in secondo luogo si è realizzata l'implementazione vera e propria del modello di PLI. A seguire è stato implementato un ambiente di generazione di scenari configurati in maniera differente tra di loro. Questo ha permesso di effettuare dei test in modo completo e eterogeneo, sperimentando i modelli in situazioni diverse a livello di numero

e modalità di generazione dei task. A fronte di questo è stata condotta una dettagliata analisi dei risultati ottenuti, in termini di numero di task accettati, guadagno e saturazione dell'infrastruttura. L'analisi è coerente con la formulazione del modello e conferma il suo corretto funzionamento a fronte di diversi scenari d'esecuzione del sistema. In aggiunta è stata realizzata una prima versione di algoritmo risolutivo, basato sulla combinazione di tecniche euristiche e di programmazione lineare. Dalle analisi effettuate l'euristica produce risultati quasi sempre accettabili in tempi nettamente inferiori rispetto al corrispettivo modello ottimale.

In conclusione la nostra ricerca ha dimostrato che la schedulazione dei task in ambito green edge computing è un problema complicato che richiede soluzioni innovative. Questo lavoro di tesi affronta il problema e fornisce un contributo importante in termini di formulazione e di implementazione, fornendo allo stesso tempo un algoritmo risolutivo per la risoluzione. I nostri risultati indicano che tecniche efficaci di scheduling possono contribuire in modo significativo alla creazione di un'infrastruttura edge più efficiente e sostenibile. Il nostro lavoro è il punto di partenza per uno studio sempre più dettagliato di scheduling in ambienti di green edge computing e lascia spazio alla creazione di nuovi algoritmi risolutivi e alla valutazione di scenari di esecuzioni più complessi.

Un possibile sviluppo futuro di questo lavoro di tesi riguarda la valutazione delle prestazioni considerando un più ampio insieme di scenari, sia considerando diverse topologie di rete, sia una diversa dinamica temporale delle risorse green, al fine di generalizzare i risultati preliminari mostrati in questo lavoro. In primo luogo va notato che i test finora condotti si basano esclusivamente su una specifica topologia di rete, definita in funzione dei dati energetici acquisiti. Tuttavia, per migliorare e generalizzare il progetto, sarebbe opportuno fornire diverse topologie di rete che differiscano tra loro per il numero di nodi, densità e altri parametri pertinenti. Un ulteriore aspetto da investigare in futuro, riguarda la modellazione più accurata e

realistica dei tempi trasmissione dei dati, considerando parametri come la banda di trasmissione disponibile, i ritardi di propagazione e la quantità di dati da trasmettere.

Per quanto riguarda i metodi di risoluzione online del problema di schedulazione dei task, si potrebbero investigare soluzioni basate su tecniche di *Machine Learning*, come in [4] dove si utilizza un approccio di *Deep Reinforcement Learning* per affrontare il problema di job scheduling. Il Deep Reinforcement Learning (DRL) rappresenta un'importante area di ricerca nell'ambito dell'apprendimento automatico e combina tecniche di reinforcement learning (apprendimento per rinforzo) con le deep neural networks. Il loro studio rivela che il metodo di DRL ha il potenziale per superare i tradizionali algoritmi di allocazione delle risorse in una varietà di ambienti complicati.

Infine si può considerare un approccio distribuito al problema affrontato in questa tesi, valutando differenze di prestazioni rispetto all'approccio centralizzato proposto. Seguendo un approccio decentralizzato ogni singolo nodo prende una decisione locale riguardo allo scheduling dei task che riceve. In particolare possono decidere se eseguire localmente il task oppure inviarlo a qualche altro nodo della rete che possa eseguire il compito per loro. Questa scelta è complicata, in quanto può dipendere da un elevato numero di fattori, come la disponibilità energetica, la conoscenza della topologia della rete o il carico dei task da processare.

## Appendice A

### File di configurazione

Di seguito è presente un esempio di file di configurazione utilizzato per generare diversi scenari di test per il modello. Possiamo notare che il file presenta una serie di parametri che permettono di personalizzare l'implementazione del modello. In primo luogo è possibile specificare la tipologia del modello stesso, che come analizzato nel capitolo 2, può essere standard o con accumulatori di energia. Successivamente il file permette di configurare il numero dei timeslot e il numero di task che il sistema deve processare nel tempo. Inoltre possiamo specificare la distribuzione di probabilità di generazione dei task: ad esempio i task di tipo *trivial*, secondo il file 1, vengono generati con il 40% in alcune configurazioni, con il 70% e 20% in altre ed infine con il 25% in altre ancora. Gli altri parametri configurabili sono la distribuzione secondo cui i task vengono generati nell'infrastruttura e quali sono i nodi che generano i task. Infine possiamo specificare i parametri di `output_size` e altri parametri energetici della rete, come il consumo energetico unitario di processamento, il costo dell'energia green, dell'energia accumulata e i costi minimi e massimi dell'energia brown.

```
1 model_type: ['standard', 'accumulator']
2 timeslots: 96
3 number_of_tasks: [500, 1000, 2000, 5000]
4 task_prob:
5     trivial: [0.4, 0.7, 0.2, 0.25]
6     expensive: [0.1, 0.1, 0.4, 0.25]
7     competitive: [0.3, 0.1, 0.1, 0.25]
8     extreme: [0.2, 0.1, 0.3, 0.25]
9
10 arrival_time:
11     distributions:
12         - uniform:
13             min_value: 0
14             max_value: timeslots
15
16 input_nodes: [[0, 10], [1, 7, 9], [6]]
17 output_size: 100
18
19 energy_consumption: 0.015
20 green_cost: 0.5
21 battery_cost: 0.8
22 min_brown_cost: 1
23 max_brown_cost: 4
```

*Listing 1:* File di configurazione

## Appendice B

### Model

Di seguito è analizzata la struttura dell'implementazione del modulo *model* (3.6) e di come **Gurobi** è stato integrato al nostro progetto per creare il modello di PLI, aggiungere i set di variabili decisionali, i vincoli e la funzione obiettivo.

#### B.1 Variabili Decisionali

Il modulo permette di creare i set di variabili decisionali definiti nei modelli del capitolo 2. Tramite la funzione del modulo è possibile definire il nome del set, upper bound, lower bound e gli indici sui quali i set si basano. La funzione utilizza a sua volta la funzione **addVars** messa a disposizione da **Gurobi**. Il seguente codice crea il set di variabili Z nel modello **m**.

---

```
Z = model.add_vars(m, 'Z', N, binary=True)
```

---

Ricordando quanto spiegato nel capitolo 2, il set presenta una variabile binaria per ogni task del sistema. La funzione infatti specifica **N** come indice del set, ossia il numero totale dei task. Essendo un set di variabili binarie e non continue, la funzione specifica il vincolo **binary=True**.

Il seguente esempio invece permette di creare il set delle X, indicizzate su **N**, **F** e

$T$ , che rappresentano rispettivamente il numero dei task, delle facilities e dei timeslots.  $X_{ij}^t$  indica la porzione di task  $i$  assegnato al nodo  $j$  durante il timeslot  $t$  e per questo è una variabile continua che appartiene all'intervallo  $[0, 1]$ . L'upper bound 1 è già definito all'interno della funzione `add_vars` del modulo per il set  $X$ .

---

```
X = model.add_vars(m, 'X', N, F, T)
```

---

## B.2 Vincoli

Il modulo contiene una serie di funzioni che consentono di specificare i vari vincoli del modello. In particolare alcune funzioni possono essere utilizzate per entrambi i modelli di ILP, mentre alcune sono create nello specifico per definire i constraints di un modello in particolare.

**Vincolo di accettazione task** Consideriamo per esempio il vincolo 2 del modello standard (2.2.2) ed il vincolo 8 del modello con accumulatori (2.3.1). I due vincoli sono identici in quanto non dipendono dalla tipologia del modello da eseguire. Il modulo *model* utilizza la stessa funzione per generare il set di vincoli per entrambi i modelli. Di seguito il vincolo del modello di PLI:

$$\sum_{j \in F} \sum_{t \geq a_i + \delta_{kj}}^{t \leq a_i + \tau_i - \delta'_{jk}} X_{ij}^t = Z_i \quad \forall i \in N \quad (26)$$

(27)

Nell'implementazione attuale il  $\delta_{jk}$  e il  $\delta'_{jk}$  sono calcolati con il così detto *taglio ad hop*. In particolare la distanza tra il nodo  $j$  ed il nodo  $k$  è calcolata come numero di hop nel cammino minimo tra  $j$  e  $k$  nella rete. In realtà, come analizzato nel capitolo 3.5, gli archi nella topologia sono pesati con la distanza in km tra i nodi della rete, che

a loro volta sono posizionati secondo le coordinate GPS delle città del Beglio che rappresentano. Come sviluppo futuro il  $\delta$  potrebbe essere calcolato in funzione della distanza tra i nodi, della velocità di trasmissione della rete e della quantità di dati da inviare da un nodo ad un altro.

**Vincolo di capacità energetica** Se consideriamo i vincoli 4 e 5 (2.2.2) ed i vincoli 11 e 12 (2.3.1) è possibile notare che seguono la stessa struttura di base. I vincoli specificano che la quantità energetica, proveniente da una determinata fonte, allocata in una facility  $j$  al tempo  $t$  non deve eccedere la quantità di energia disponibile in quella facility a quel timeslot per quella specifica fonte di alimentazione. Anche in questo caso il modulo mette a disposizione un'unica funzione generale per generare i constraints dei modelli. Di seguito i vincoli dei modelli di PLI:

$$G_j^t \leq \overline{G}_j^t \quad \forall j \in F, \forall t \in T \quad (28)$$

$$B_j^t \leq \overline{B}_j^t \quad \forall j \in F, \forall t \in T \quad (29)$$

$$H_j^t \leq \overline{H}_j^t \quad \forall j \in F, \forall t \in T \quad (30)$$

$$A_j^t \leq \widetilde{A}_j^t \quad \forall j \in F, \forall t \in T \quad (31)$$

**Vincolo di capacità nelle facilities** Se le funzioni analizzate precedentemente vengono condivise da entrambi i modelli, lo stesso non vale per i seguenti vincoli. I due constraints che esprimono il vincolo di capacità per ogni facility nei rispettivi modelli sono il (3) e il (9). Sebbene seguano una struttura simile, sono stati implementati con due funzioni differenti. Ricordando il capitolo 2, i due vincoli specificano che il carico di lavoro totale per ogni facility in ogni timeslot non deve eccedere la capacità energetica della facility nel timeslot, calcolata come somma delle capacità delle varie fonti energetiche. Di seguito il vincolo del modello standard e il vincolo del modello con accumulatori:

$$\sum_{i \in N} \mu_i X_{ij}^t d_i \leq G_j^t + B_j^t \quad \forall j \in F, \forall t \in T \quad (32)$$

$$\sum_{i \in N} \mu_i X_{ij}^t d_i \leq H_j^t + A_j^t + B_j^t \quad \forall j \in F, \forall t \in T \quad (33)$$

(34)

Nell’implementazione dei vincoli la parte sinistra della disequazione viene moltiplicata per *energy\_consumption*/60. Questo per confrontare correttamente la parte sinistra e destra della disequazione. Mentre la sommatoria di sinistra è misurata in KB, la parte destra esprime un valore in MW. Come anticipato nel capitolo 2 la variabile *energy consumption* (J/KB) esprime la quantità di joule necessari per processare un KB. La variabile moltiplica la sommatoria di sinistra, evidenziando la quantità di joule necessari per processare il carico di lavoro della facility. A questo punto, ricordando che un watt equivale ad un joule al secondo, la parte sinistra viene divisa per 60 per ottenere la stessa unità di misura da entrambe le parti.

**Vincolo di capacità energetica totale** Il seguente vincolo, così come i successivi, è implementato esclusivamente per il modello con accumulatori. Il vincolo specifica che la somma delle variabili decisionali energetiche non deve eccedere la capacità totale della facility.

$$H_j^t + A_j^t + B_j^t \leq \bar{C}_j \quad \forall j \in F, \forall t \in T \quad (35)$$

(36)

**Vincolo di ricarica degli accumulatori** Il vincolo sulla ricarica è implementato utilizzando la funzione `min_` di **Gurobi**, che imposta una variabile decisionale al minimo di un elenco di variabili decisionali o costanti. L’implementazione permette quindi

di aggiornare le variabili decisionali con un valore pari ad una potenziale ricarica della batteria oppure pari alla capacità massima della batteria.

$$\widetilde{A}_j^t = \min\{\widetilde{A}_j^{t-1} - A_j^{t-1} + \overline{H}_j^{t-1} - H_j^{t-1}, \overline{A}_j\} \quad \forall j \in F, \forall t \in T \quad (37)$$

(38)

**Vincolo di capacità iniziale degli accumulatori** L'ultima implementazione forza semplicemente il livello di tutte le batterie a zero all'inizio della simulazione.

$$\widetilde{A}_j^0 = 0 \quad \forall j \in F \quad (39)$$

### B.3 Funzione Obiettivo

Anche per la funzione obiettivo, il modulo dispone di due diverse implementazioni per i rispettivi modelli, sebbene seguano la stessa struttura di base. Le due funzioni utilizzano a loro volta la funzione `setObjective` di Gurobi.

## Appendice C

### Utils

Di seguito è esaminata l'implementazione del modulo *utils*, che contiene tutte le funzioni di import/export ed inoltre permette di generare le facilities e i task nel sistema. Il modulo è utilizzato da entrambe le implementazioni dei modelli di PLI.

#### C.1 Generazione facilities

Il modulo permette inoltre la creazione delle facilities all'interno dell'infrastruttura. Sebbene la topologia della rete è già definita nel modulo *topology* (3.5), il modulo *utils* definisce per ogni facility le quantità energetiche ed i prezzi computazionali nel tempo. In particolare il modulo presenta due funzioni, una per ogni tipologia di modello, in quanto a seconda della tipologia cambiano le fonti di alimentazione energetiche. Di conseguenza la funzione per il modello standard assegna ad ogni nodo una quantità di energia brown e green per ogni timeslot ed il loro rispettivo prezzo. La funzione per il modello con accumulatori assegna ad ogni nodo una quantità di energia harvesting ed una capacità totale energetica per la facility. La quantità di energia nella batteria è definita da una variabile decisionale del modello, mentre la quantità di energia brown non viene specificata, in quanto comunque il sistema è limitato nel suo uso dal vincolo (10) del modello.

## C.2 Generazione task

Il modulo mette a disposizione una funzione per la generazione dei task nell'infrastruttura. La funzione crea N task seguendo la distribuzione definita nel file di configurazione 1 per definire gli `arrival_times`. Come analizzato in precedenza sono state definite quattro tipologie di task, differenti l'una dall'altra per `reward`, `cpu_cycle_cost` e `deadline`. La funzione assegna ogni task ad una certa tipologia, seguendo la probabilità definita sempre nel file di configurazione. Di seguito, in Figura 27, un esempio di task generato dalla funzione.

```
1: {'#': 1,
    'I': 2000,
    'O': 100,
    'a': 76,
    'd': 5,
    'k': 0,
    'r': 10,
    'type': 'competitive',
    'u': 3},
```

**Figura 27:** Esempio di task

## C.3 Import

Le funzioni di import sono condivise da entrambe le tipologie di modello. Le funzioni si occupano di leggere i dati energetici e i dati sulla domanda energetica processati in precedenza (3.4).

## C.4 Export

Il modulo presenta una serie di funzioni di export dei risultati, che verranno poi analizzati nel capitolo 5. In particolare i risultati sono organizzati in una cartella di output, nella quale ogni scenario presenta una cartella identificata con l'`id` dello scenario. Per ogni scenario vengono memorizzati il risultato della funzione obiettivo ed i valori delle variabili decisionali del modello. Vengono salvati inoltre i costi energetici delle varie fonti e le capacità energetiche di ogni facility. Infine le funzioni permettono di memorizzare il dettaglio dei task generati dal sistema e il livello di pienezza delle facilities nel tempo.

## Bibliografia

- [1] F. Spinelli, A. Bazco-Nogueras, and V. Mancuso, “Edge gaming: A greening perspective,” *Computer Communications*, vol. 192, pp. 89–105, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0140366422001748>
- [2] H. Ma, P. Huang, Z. Zhou, X. Zhang, and X. Chen, “Greenedge: Joint green energy scheduling and dynamic task offloading in multi-tier edge computing systems,” *IEEE Transactions on Vehicular Technology*, vol. 71, no. 4, pp. 4322–4335, 2022.
- [3] M. J. Neely, *Stochastic network optimization with application to communication and queueing systems*, 1st ed., 2010, vol. 3.
- [4] W. Chen, Y. Xu and X. Wu, “Deep reinforcement learning for multi-resource multi-machine job scheduling,” November 2017. [Online]. Available: <https://arxiv.org/abs/1711.07440>
- [5] ITU-T Study Group, “Consideration on 5G transport network reference architecture and bandwidth requirements,” ITU-T, Tech. Rep., 2018.

- [6] ETSI MEC Leadership Team, “ETSI MEC: An Introduction,” December 2022. [Online]. Available: [https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/ETSI-MEC-Public-Overview\\_Generic.pdf](https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/ETSI-MEC-Public-Overview_Generic.pdf)
- [7] K. Zhang, S. Leng, Y. He, S. Maharjan, and Y. Zhang, “Mobile edge computing and networking for green and low-latency internet of things,” *IEEE Communications Magazine*, vol. 56, no. 5, pp. 39–45, 2018.
- [8] M. Caramia, S. Giordani, F. Guerriero, R. Musmanno, D. Pacciarelli, *Ricerca Operativa*. ISEDI, 2014.
- [9] Elia Group, “Photovoltaic power production estimation and forecast on Belgian grid (Historical).” [Online]. Available: <https://opendata.elia.be/explore/dataset/ods032/table>
- [10] ——, “Wind power production estimation and forecast on Belgian grid (Historical).” [Online]. Available: <https://opendata.elia.be/explore/dataset/ods031/table>
- [11] Terna - Rete Elettrica Nazionale S.p.A., “Market Load.” [Online]. Available: <https://www.terna.it/en/electric-system/transparency-report/market-load>