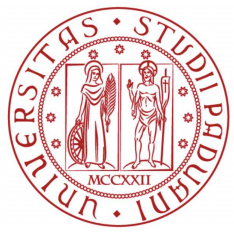


UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA TRIENNALE IN
INGEGNERIA BIOMEDICA

Integrazione di parametri da dateset biologici in un simulatore per comunità microbiche basato su modello multi-agente

Relatore:
ING. MASSIMO BELLATO, PHD

Laureando:
NICCOLÒ VENTURINI DEGLI ESPOSTI
MATR. 2001504

Correlatrice:
CHIAR.MA PROF.SSA BARBARA DI CAMILLO

Anno Accademico 2022/2023

Data di Laurea 28/09/2023

A nonno Aurelio

Abstract

Le comunità microbiche rivestono un ruolo fondamentale in svariati campi, come ad esempio l'insieme dei microorganismi che convivono all'interno dell'intestino umano (gut microbiota) supportando ad esempio l'attività digestiva. Lo studio delle comunità microbiche è in continua evoluzione ma ancora non si è giunti ad avere un quadro completo sulle interazioni tra le comunità e l'ambiente circostante, soprattutto a causa della complessità della coltura e studio di alcune specie di microrganismi in laboratorio. Per questo, lo sviluppo di metodi computazionali che permettano di predire l'evoluzione di specie batteriche coesistenti in risposta a stimoli esterni, risulta di grande interesse nella comunità scientifica. Con questa idea è stato sviluppato un simulatore di comunità microbiche, scritto in linguaggio Python, basato su un modello Agent Based (ABM) finalizzato a mimare l'interazione di agenti autonomi da cui ottenere l'evoluzione del sistema complessivo.

Lo scopo di questo elaborato è ottenere una panoramica sulla reperibilità di dati biologici di specie batteriche e terreni di coltura, al fine di sviluppare un framework per l'integrazione degli stessi nel simulatore. Questo col fine ultimo di rendere le simulazioni quantitativamente realistiche. Per fare ciò sono stati studiati vari database, i principali sono: *BacDive, MediaDive, Agorà, AGREDA*.

Dato il grandissimo numero di reazioni e metaboliti presenti nei metabolismi dei microorganismi non è immediato trovare i dati utili necessari; questo ha richiesto lo studio approfondito dei database, in modo tale da capire quali dati fossero effettivamente utilizzabili.

Ulteriori sviluppi futuri potranno essere:

- Ampliare la caratterizzazione spaziale, così da riuscire ad effettuare delle simulazioni in due dimensioni (ad esempio simulando così la crescita di batteri in una piastra Petri) e possibilmente anche in strutture tridimensionali.
- L'implementazione di un flusso dinamico di metaboliti (ad esempio simulando un tratto intestinale con un bolo in entrata).
- La validazione dei risultati ottenuti tramite test in laboratorio su sotto-insiemi di batteri e metaboliti gestibili manualmente.

Indice

Capitolo 1

Simulatore per comunità microbiche multi-agente

Questo elaborato di tesi vuole continuare un progetto più ampio, denominato *BactLife*, il cui scopo è progettare un simulatore per comunità microbiche basato su un modello multi-agente, continuando i lavori presentati nelle tesi triennali in Ingegneria Biomedica: ”Implementazione di un simulatore per comunità microbiche basato su un modello multi-agente” di A. Calzavara [1], ”Bactlife: simulatore per comunità batteriche - sviluppo del pacchetto Python” di A. Lucchiari [2], ”Bactlife: simulatore per comunità batteriche - sviluppo di interfaccia grafica in Dash” di S. Rebecca [3]. In questi tre elaborati è stata creata una prima versione del simulatore predittivo modulare e in grado di riprodurre l'evoluzione di specie batteriche in funzione dell'ambiente in cui essi si trovano e anche del loro metabolismo. É stata standardizzata la struttura ed è stata creata una *repository* su GitLab, essenziale per sviluppi futuri dato che potranno essere portati avanti su file organizzati in modo standardizzato. Infine è stata creata un'interfaccia *user-friendly*, grazie all'utilizzo di alcune librerie di Python (*Pandas* e *Plotly*), che permetta, anche agli utenti con limitate competenze informatiche, di poter eseguire una simulazione.

In questo primo capitolo verranno esposte l'importanza che hanno le simulazioni computazionali nel comprendere la crescita di comunità microbiche, le caratteristiche e le scelte implementative che hanno permesso al simulatore di essere modulare e con un'interfaccia grafica *user-friendly*.

Verrà quindi illustrato il modello utilizzato per descrivere le interazioni tra i microorganismi presenti nell'ambiente preso in considerazione, presentando le classi, le funzioni

e le variabili utilizzate, come implementati nei lavori antecedenti a questo [1–3].

1.1 Valenza Biologica

Le comunità microbiche sono composte da varie specie di batteri che interagiscono tra di loro nello stesso ambiente. Il tipo ed il numero di microbi che compongono una comunità ad un certo istante temporale dipendono quindi dalla composizione dell'ambiente stesso e da come le specie ospitate riescano o meno a sfruttare tali substrati metabolici.

Queste comunità rappresentano la maggioranza della biodiversità presenti sulla Terra e svolgono diversi ruoli essenziali, come ad esempio la decomposizione di materiale organico [4]. Un altro importante esempio è rappresentato dai vari batteri che vivono all'interno del nostro tratto gastro-intestinale, costituendo il così detto microbiota intestinale umano (*human gut microbiome - HGM*). Questi batteri hanno un'ampia gamma di vie metaboliche e rappresentano un fattore essenziale nel processo di digestione per l'ospite in cui si trovano. Dato che l'HGM è un fattore chiave nel modellare il profilo biochimico della dieta è di estrema importanza per la salute dell'ospite [5].

Tuttavia le nostre conoscenze sono ancora molto limitate, ciò rende difficile lo studio delle interazione in ambienti complessi, come potrebbe essere il tratto intestinale.

Riuscire a sviluppare un simulatore in grado di predire lo sviluppo di varie specie in uno stesso ambiente porterebbe grossi vantaggi, ad esempio renderebbe possibile evitare di effettuare alcune sperimentazioni in laboratorio, in particolar modo per le specie più pericolose e più difficili da gestire, riuscendo così a risparmiare sia risorse sia tempo, inoltre, potrebbe anche aiutare a comprendere meglio l'interazione di differenti specie batteriche che convivono nello stesso ambiente.

1.2 Reperibilità dei dati

Una grande problematica da affrontare nell'implementazione di un simulatore di comunità batteriche è riuscire a ricavare i corretti dati sui metabolismi dei batteri. Questo vuol dire saper descrivere quali metaboliti siano processati e prodotti da ogni specie batterica e quanto il loro metabolismo incrementi od ostacoli la crescita batterica stessa. Sarebbe di notevole importanza riuscire a descrivere in modo dettagliato la composizione biochimica dell'ambiente in cui si trovano, così da poter capire come sarà l'effettiva evoluzione delle comunità prese in considerazione.

Dato l'elevato numero di reazioni presenti nei metabolismi dei microorganismi, non risulta facile l'analisi di questa grande quantità di informazioni, pertanto è necessario capire quali siano i metaboliti che effettivamente vengono utilizzati e prodotti, senza dover considerare gli intermedi di reazione. Nei database contenenti i dati di interesse non sono presenti in modo dettagliato i bilanci metabolici (ossia quanta energia o che prodotti vengano generati a valle del metabolismo di un certo nutriente). È stato necessario processare i dati individuati, così da avere una stima più realistica possibile dei metaboliti necessari per la crescita dei batteri considerati e quelli che vengono prodotti dalle vie metaboliche attivate.

Sarebbe utile conoscere l'esatta composizione dei principali terreni di coltura, così da poter stimare al meglio la crescita delle specie ivi presenti. Tuttavia, come è illustrato nel Capitolo 3, non è facile avere un dato preciso sui terreni dato che alcuni dei loro componenti derivano da scarti di lavorazione e quindi non è esattamente noto cosa sia presente al loro interno.

L'obiettivo principale di questo elaborato è quello di esplorare le fonti disponibili per reperire dati riguardanti i metabolismi dei batteri e la composizione di terreni di coltura. Per elaborare questo studio sono stati presi in considerazione diversi database; tra tutti, i principali (in termini di abbondanza e qualità del dato), su cui si è poi deciso di approfondire l'attività di ricerca, sono: BacDive, MediaDive, AGORA, AGREDA [6 - 9]. L'implementazione di una struttura di dati, basata su questi database da fornire al tool *Bactlife*, dovrebbe consentire al simulatore di effettuare delle simulazioni che possano avere una vera e propria valenza - nonchè validazione - scientifica.

1.3 Il simulatore BactLife

1.3.1 Introduzione al modello

Inizialmente è stata studiata la scelta del *framework* modellistico su cui basare il simulatore. Le scelte possibili erano state (rappresentate nella figura 1.1):

- modelli meccanicistici
- modelli empirici
- modelli integrati

I modelli meccanicistici hanno una struttura matematica basata su rigorose ipotesi biologiche ed ogni parametro ha una diretta interpretazione fisiologica, i modelli empirici

hanno una struttura matematica che permette di spiegare dati empirici senza la necessità di rifarsi obbligatoriamente a strutture e parametri con una diretta interpretazione fisiologica ed infine i modelli integrati sono una combinazione di questi due modelli.

I vari modelli si possono anche classificare in base all'obiettivo del simulatore, in questo caso si dividono in:

- modelli ecologici;
- modelli microbo-effettore;
- modelli Agent Based (ABM) [10].

I modelli ecologici sono volti a predire ed analizzare le dinamiche delle popolazioni, includendo sia i cambiamenti nel tempo delle quantità delle singole popolazioni sia come si influenzano tra di loro, senza prendere in considerazioni le interazioni molecolari che vi stanno alla base (un esempio sono i modelli generalizzati di Lodka-Volterra [11 12] ed i modelli *data-driven* [13-15]).

L'obiettivo dei modelli microbo-effettore è quello di capire e descrivere quale sia il ruolo degli effettori nella crescita delle varie popolazioni batteriche. Possono essere basati su equazioni differenziali ordinarie (ODE) utilizzate per analizzare la variazione della crescita delle varie specie batteriche in funzione della concentrazione extracellulare di determinati effettori chiave. Un altro modello di questo tipo sono i *genome-scale models* (GEMs); questo tipo di modello prova a predire ogni reazione metabolica di un microorganismo grazie all'informazione genomica arricchita da evidenze sperimentali e si basa sull'analisi del bilancio dei flussi (FBA).

L'idea che sta alla base nei modelli Agent-Based (ABM) è che i sistemi possano essere modellati a partire da delle entità, che prendono il nome di agenti, aventi caratteristiche ben precise. Gli agenti interagiscono seguendo delle regole che definiscono come si comportano sia con gli altri agenti che con l'ambiente.

Per il simulatore è stato scelto di utilizzare un modello Agent-Based, in cui gli agenti sono rappresentati dalle varie specie microbiche presenti; questo è l'unico dei modelli che permette di definire le interazioni tra le varie popolazioni in funzione delle caratteristiche degli agenti e non da ipotesi fatte a monte su co-occorrenza o mutua inibizione. Questo modello è di facile implementazione dato che si può idealizzare un'agente come un oggetto definendone il comportamento tramite i propri attributi e metodi. Questi ultimi, rappresentandone di fatto le regole di comportamento, possono essere facilmente aggiornati o modificati a livello globale, senza andare a ricostituire

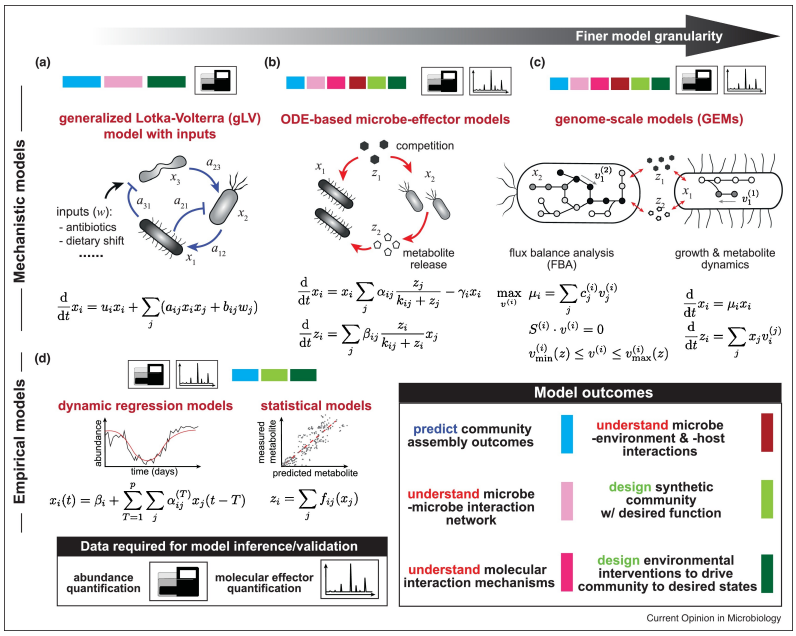


Figura 1.1: Rappresentazione schematica dei diversi modelli, immagine adattata da [10]

l'intera struttura del simulatore, rendendo l'implementazione estremamente modulabile e customizzabile, nonché di facile aggiornamento laddove la biologia molecolare e le discipline omiche possano fornire parametri più precisi e dettagliati.

1.3.2 Implementazione del simulatore

Il simulatore ad ora considera un unico spazio unidimensionale sequenziale, dove i vari agenti - ossia le specie batteriche - possono evolversi in base all'ambiente circostante, secondo un processo iterativo. Questo ambiente lineare a celle successive è stato ideato pensando ad un tratto di intestino, così da poter simulare la colonizzazione da parte della flora batterica da un tratto all'altro. Ogni specie batterica presente ha un proprio nome identificativo, tasso di crescita massimo e metabolismo generato in modo casuale mediante la funzione *RandomBacts*.

Nell'esempio di riferimento che verrà analizzato nei prossimi paragrafi, il sistema rappresentato è costituito da 5 celle all'interno delle quali è inizializzata una quantità casuale di un numero di ingredienti complessivo pari a 25, colonizzate da 5 specie batteriche differenti.

Nella prima fase della simulazione vengono inizializzati i vari dati che sono definiti in modo casuale dal numero di nutrienti in ogni cella e dalla quantità di batteri per ogni specie, questi ultimi sono presenti solo nella prima cella. Le quattro celle successive inizialmente sono senza specie batteriche e verranno popolate man mano che le specie

aumenteranno di numero e migreranno ordinatamente verso la cella successiva per simulare un flusso peristaltico. Ad ogni simulazione sono generati e memorizzati dei valori *seed*, così che la simulazione sia sì casuale ma anche riproducibile.

Successivamente è presente una fase dinamica, implementata come un processo iterativo dove ogni simulazione corrisponde ad un'ora di evoluzione delle comunità batteriche. Ad ogni ciclo vengono calcolati per ogni specie in una determinata cella:

- la quantità di nuovi batteri nati, di cui la metà verranno spostati nella cella successiva, considerando il fattore di crescita che dipende dal tasso di crescita massimo e dai nutrienti mangiati;
- la quantità di batteri morti che dipende dal tasso di tossicità dei metaboliti presenti sommato al tasso di morte basale;
- il numero di batteri che si sposteranno nella cella successiva, è impostato al 5% del numero totale di batteri presenti nella cella presa in considerazione, senza considerare i nuovi nati e, nel caso in cui si sia nell'ultima cella, i batteri in questione verranno espulsi dall'ambiente.

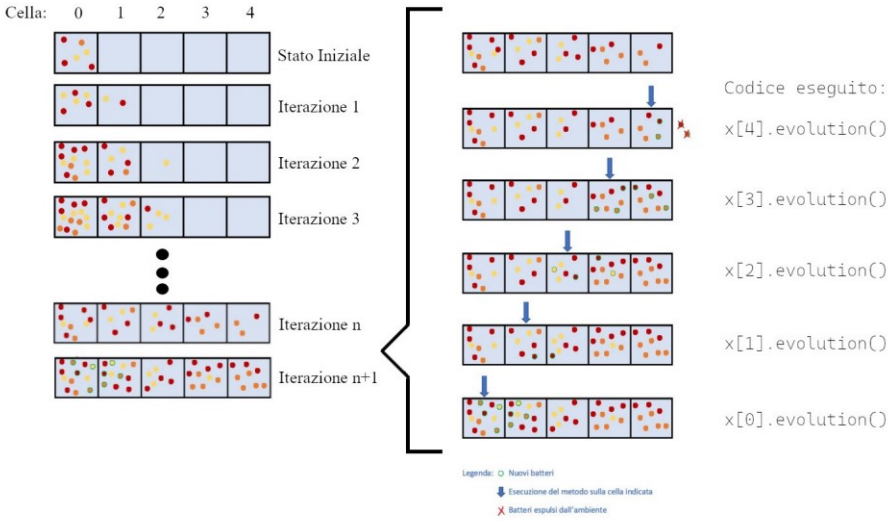


Figura 1.2: A sinistra sono rappresentate n+1 iterazioni, mentre a destra sono schematizzate le modifiche apportate dalla funzione *evolution* per ogni cella in una specifica iterazione.

1.3.3 Classi e Funzioni

In questo paragrafo verranno illustrate le varie classi e funzioni che hanno permesso di definire il modello precedentemente descritto; il linguaggio adottato è Python. In figura

1.3 è riportato uno schema UML in cui sono riassunte le classi e i loro metodi. Sono state riportate anche le funzioni accessorie che non appartengono a nessuna classe.

Nell'implementazione del simulatore è stato scelto di utilizzare due classi:

Classe Cell

La classe *Cell* rappresenta una cella, che è l'ambiente che conterrà i metaboliti e i batteri. I suoi attributi e metodi sono riportati nella tabella 1.1. Le dimensioni degli attributi indicano: *"s"* il numero di specie batteriche, *"n"* il numero di diversi nutrienti, *"i"* il numero di iterazioni ed infine *"c"* è il numero di celle di cui è composto il vettore.

	Nome	**Descrizione**	**Dimensione**
attributi	*_food*	vettore dei nutrienti, una lista contenente in ogni posizione un intero che indica la quantità di quel nutriente nella cella	n
	_MatrFood	matrice che tiene conto dell'andamento dei nutrienti nel tempo, ogni riga rappresenta il vettore *_food* ad una data iterazione	n x i
	_bact	dizionario che ha come chiavi istanze della classe Bact, mentre i valori sono numeri interi che indicano la quantità di batteri di quella classe presenti nella cella	s
	_x	lista contenente le celle usate nella simulazione	c
	_pos	intero che indica la posizione nel vettore delle celle	-
	getMatrFood, *getFood*, *getBact*	metodi di accesso per: la matrice dei nutrienti, il vettore *_food* ed il dizionario *_bact*	

metodi	*addBact*	metodo *mutator* che permette di aggiornare il dizionario *_bact*, sommandogli un nuovo dizionario costruito allo stesso modo contenente i nuovi batteri da aggiungere a quelli presenti in seguito all'evoluzione del sistema	
	food_upd	aggiorna il vettore dei nutrienti *_food*	
	death	calcola il numero di microbi morti all'iterazione corrente per ogni specie	
	evolution	gestisce interamente la fase dinamica del simulatore, calcolando lo spostamento, la morte, la riproduzione e l'interazione coi nutrienti presenti nella cella considerata	

Tabella 1.1: Attributi e metodi della classe Cell

Classe Bact

La classe *Bact* rappresenta l'agente, che è una determinata specie batterica in una cella. I suoi attributi e metodi sono rappresentati nella tabella 1.2. Le dimensioni degli attributi indicano: *"s"* il numero di specie batteriche, *"n"* il numero di diversi nutrienti.

	Nome	**Descrizione**	**Dimensione**
attributi	*_species*	attributo di classe; una lista di stringhe ciascuna delle quali definisce una specifica specie batterica	s
	_type	stringa tra quelle in *_species*, che definisce la specie	-
	_m	lista di interi che definisce il metabolismo della specie batterica rispetto ai metaboliti presenti, dove: +1 prodotto, 0 ignorato, -1 consumato	n
	_t	vettore di 1 e 0 che rappresenta la possibile tossicità di un nutriente nei confronti di una determinata specie batterica, dove: 1 indica nutriente tossico, 0 nutriente non tossico	n

	_maxGr	float che indica il tasso di crescita massimo	-
	_maxTox	float che indica il tasso di tossicità massimo	-
	_pos	posizione del batterio all'interno del vettore elementi di *Cell*	-
metodi	*getm*, *gett*, *getMaxGr*, *getpos*	metodi di accesso che restituiscono: il vettore del metabolismo, il vettore di tossicità, il coefficiente di crescita massimo e la posizione	
	getgrowth	calcola il fattore di crescita di una specie in una determinata cella	
	getTox	calcola il coefficiente di tossicità di una specie in base ai nutrienti presenti in una cella	

Tabella 1.2: Attributi e metodi della classe Bact

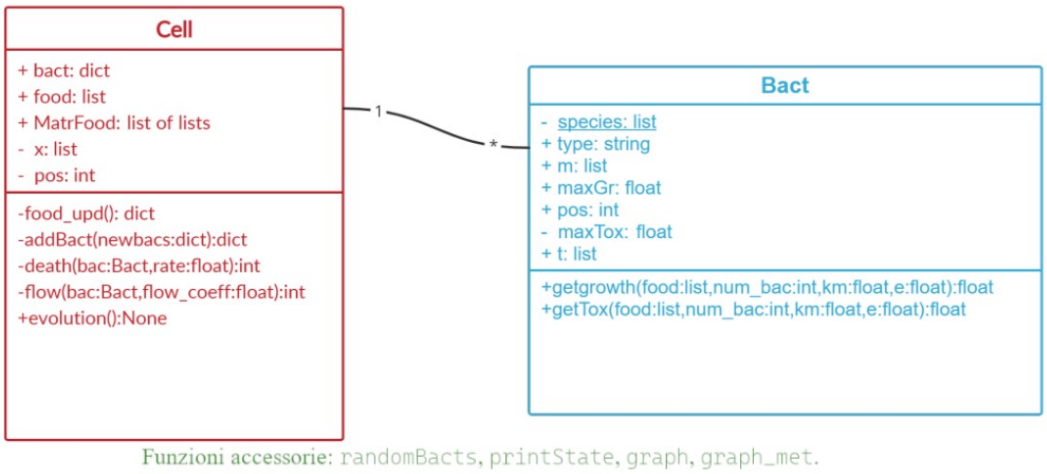


Figura 1.3: Diagramma UML delle classi, sono presenti anche le funzioni accessorie che non appartengono a nessuna classe

1.3.4 Sviluppo del pacchetto

Uno strumento fondamentale per lo sviluppo del pacchetto è stato l'utilizzo di GIT, un software volto alla gestione di *repository*, che fornisce un sistema di controllo di versione, indipendentemente dal linguaggio di programmazione che si sta utilizzando. Git permette di tenere traccia delle modifiche effettuate al progetto ed eventualmente

di tornare indietro ad una versione precedente. É data la possibilità di lavorare parallelamente grazie all'utilizzo di *branch*. Gli sviluppatori, per poter utilizzare questo strumento, possono fare riferimento a piattaforme online come GitLab o GitHub.

Il primo passo della realizzazione del pacchetto è stato definire una struttura gerarchica di BactLife. Partendo dalla prima versione del codice implementato nella tesi di A. Calzavara [1], sono state divise le classi e le funzioni in vari file (i moduli di BactLife). La spiegazione dettagliata dello sviluppo del pacchetto è riportata nella tesi di A. Lucchiari [2]. I criteri di suddivisione sono essenzialmente due:

- assegnare a differenti moduli le classi e le funzioni che hanno differenti scopi;
- facilitare l'importazione ed il richiamo degli attributi assegnandoli nomi semplici ed evocativi.

In figura 1.4 è rappresentata la struttura del pacchetto, i criteri sopra descritti servono per rendere il codice comprensibile anche per coloro che non abbiano effettivamente lavorato al programma. Questo è fondamentale per aiutare il lavoro dei futuri sviluppatori. Per fare ciò sono state apportate delle modifiche rispetto alla versione implementata con la prima tesi [1].

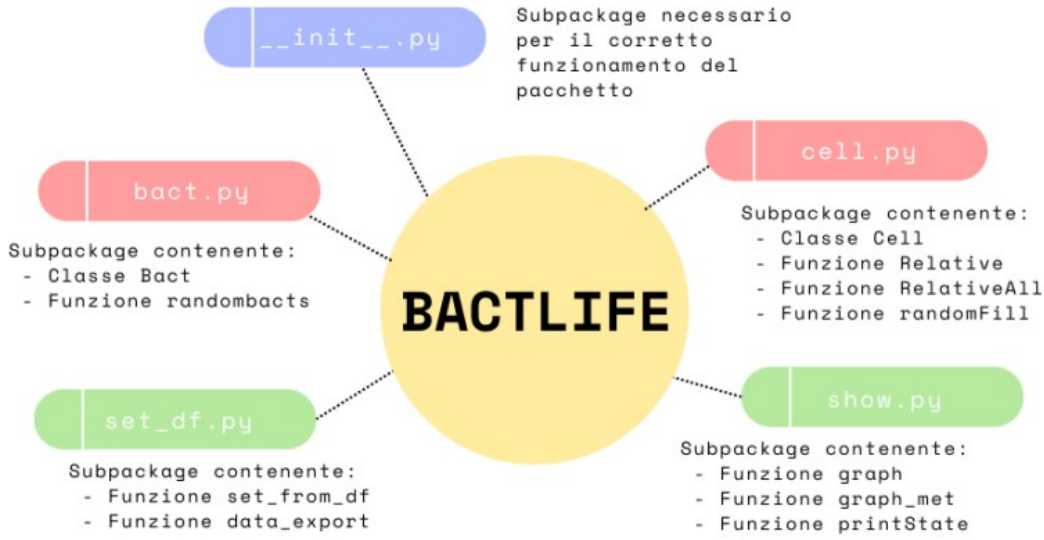


Figura 1.4: Schema dei *subpackage* di BactLife

Nel file **bact.py** sono stati inseriti gli attributi che gestiscono la creazione e la caratterizzazione di ogni specie batterica presente, che sarebbero la classe *bact* ed il metodo *_randomBacts* utile per definire in modo randomico le interazioni delle varie specie coi nutrienti. Nella classe *bact* sono state effettuate delle modifiche inserendo i metodi

set_m e *set_t*, utili per la modifica della specie batterica, ed è stato rimosso il metodo *getGrowth*.

Nel file **cell.py** sono presenti le classi e le funzioni necessarie alla gestione delle celle, le componenti del vettore in cui viene eseguita la simulazione e i dati di ogni specie batterica che popola ogni cella. Sono presenti la classe *cell* e le funzioni *Relative*, *RelativeAll* e *RandomFill*. Le prime due funzioni hanno il ruolo di generare dei dizionari le cui chiavi sono i nomi delle specie batteriche e i valori sono le percentuali delle abbondanze delle specie batteriche. La prima funzione ha come valori il risultato della divisione tra il numero di batteri di una specie in una determinata cella per il numero totale di batteri presenti in quella cella. I valori dei dizionari che vengono creati dalla seconda funzione sono il rapporto tra il numero di batteri di una determinata specie rispetto ai batteri totali, considerando la totalità delle celle. La terza funzione crea una lista di lunghezza pari al numero di celle i cui valori sono numeri interi compresi tra due numeri standard. Rispetto alla versione precedente questa implementazione ha portato ad avere nuove e più complete funzionalità all'interno del simulatore. Sono stati aggiunti degli attributi e i loro metodi di accesso:

- *_MatrGrowth*, lista dei coefficienti di crescita ottenuti ad ogni iterazione, col rispettivo metodo di accesso *getMatrGrowth*;
- *_bactCell*, lista di dizionari contenente le percentuali di ogni specie batterica rispetto al numero totali di batteri presenti nella cella considerata, col relativo metodo di accesso *getBactCell*;
- *_bactAll*, lista di dizionari contenente le percentuali di ogni specie batterica rispetto al numero totale di batteri presenti nell'intero vettore delle celle, col relativo metodo di accesso *getBactAll*.

In questa classe è stato aggiunto il metodo *getGrowth*, precedentemente implementato nella classe *Bact*. Il metodo *RelUpd*, la cui funzione è quella di combinare insieme le funzioni *Relative* e *RelativeAll*, permette di ottenere dizionari con valori relativi alle specie presenti in una cella, normalizzati rispetto ad una sola di esse oppure rispetto al loro totale, andando così a comporre *_bactCell* e *_bactAll*.

Il modulo **show.py** ha al suo interno le funzioni necessarie a presentare i dati in modo grafico, che consistono in *graph*, *graph_met*, *printState*.

Il modulo **set_df.py** permette l'interazione tra l'interfaccia grafica (esposta nel paragrafo 1.3.5) e l'algoritmo del simulatore grazie all'utilizzo di *dataframe*. In questo file sono state implementate due funzioni, *set_from_df* avente lo scopo di dare la possibi-

lità all'utente di modificare le caratteristiche delle specie batteriche interagendo solo con l'interfaccia grafica e *data_export* che crea un file excel contenente i dati necessari all'inizializzazione della simulazione.

Sono stati aggiunti dei file accessori il cui scopo è rendere distribuibile il pacchetto, avere un'efficiente interconnessione tra il codice di esecuzione e l'elaborazione dei dati del simulatore (*back-end*) ed interfaccia grafica (*front-end*), che sono:

- **__init.py__** deve essere contenuto all'interno di ogni cartella del pacchetto. É un file necessario per eseguire il codice, che contiene le istruzioni indispensabili all'importazione del pacchetto. Il file *__init.py__* è fondamentale per comunicare che la cartella in cui si trova dovrà essere considerata *package*, ossia un modo per organizzare il codice gerarchicamente secondo delle cartelle e dei file, permettendo di rendere il progetto ordinato e modulare. La divisione secondo *package* è riconosciuta dalla presenza del file *__init.py__* che viene eseguito quando è importato un file del *package*.
- **setup.py**, in questo file viene importata ed eseguita la funzione *setup*, contenuta nel pacchetto *setuptools*. Ai parametri di questa funzione vengono attribuiti le specifiche tecniche del pacchetto, come: nome, versione, licenza, descrizione e pacchetti richiesti.
- **__all__** e **__name__**, il primo file è una lista rappresentante gli attributi scaricati a seguito dell'esecuzione del comando "*from packagename import **"*, mentre il secondo è una stringa che rappresenta il nome del modulo. Nel caso in cui il modulo non sia importato ma eseguito come programma principale a '*__name__*' viene assegnata la stringa '*__main__*'. Questa caratteristica serve per poter testare il modulo.

Nella tabella 1.3 sono state riportate in modo schematico le variabili implementate nella tesi di A. Lucchiari [2] dovute allo sviluppo del pacchetto. Le dimensioni degli attributi indicano: "*s*" il numero di specie batteriche, "*n*" il numero di diversi nutrienti, "*i*" il numero di iterazioni e infine "*c*" è il numero di celle di cui è composto il vettore.

Classe di appartenenza (modulo)	**Nome**	**Descrizione**	**Dimensione**
	_species	attributo di classe; una lista di stringhe ciascuna delle quali definisce una specifica specie batterica	s

Bact (in bact.py)	*_type*	stringa tra quelle in *_species*, che definisce la specie	-
	_m	lista di interi che definisce il metabolismo della specie batterica rispetto ai metaboliti presenti. Dove: +1 prodotto, 0 ignorato, -1 consumato	n
	_t	vettore di 1 e 0 che rappresenta la possibile tossicità di un nutriente nei confronti di una determinata specie batterica. Dove: 1 indica nutriente tossico, 0 nutriente non tossico	n
	_maxGr	float che indica il tasso di crescita massimo	-
	_maxTox	float che indica il tasso di tossicità massimo	-
	_pos	posizione del batterio all'interno del vettore elementi di *Cell*	-
Cell (in cell.py)	*_food*	vettore dei nutrienti, una lista contenente in ogni posizioni un intero che indica la quantità di quel nutriente nella cella	n
	_MatrFood	matrice che tiene conto dell'andamento dei nutrienti nel tempo, ogni riga rappresenta il vettore *_food* in una data iterazione	n x i
	_MatrGrowth	matrice che contiene i coefficienti di crescita assunti nelle varie iterazioni dalle singole specie batteriche presenti all'interno della cella. Una specifica riga rappresenta i coefficienti di crescita di una specie batterica in una determinata iterazione	i x s

	_bact	dizionario che ha come chiavi istanze della classe Bact, mentre i valori sono numeri interi che indicano la quantità di batteri presenti nella cella	s
	_bactCell	dizionario *_bact* normalizzato rispetto alla totalità di batteri presenti nella cella considerata	s
	_bactAll	dizionario *_bact* normalizzato rispetto alla totalità di batteri presenti nel vettore celle	s
	_x	lista contenente le celle usate nella simulazione	c
	_pos	intero che indica la posizione nel vettore delle celle	-

Tabella 1.3: Attributi implementati dovuti allo sviluppo del pacchetto

1.3.5 Sviluppo dell'interfaccia grafica in Dash

La Graphical User Interface (GUI) è l'interfaccia grafica che permette di riprodurre il codice back-end, rendendo possibile all'utente l'interazione col computer. Questa sezione è stata sviluppata nella tesi di Sara Rebecca [3]. Questo rende possibile, anche ad un'utente privo di una elevata conoscenza informatica, l'utilizzo del simulatore. In particolare l'interfaccia grafica è necessaria per poter modificare l'impostazione dei parametri iniziali di una simulazione, senza dover interagire col codice in Python. Ciò che l'utente è in grado di modificare con l'interfaccia grafica è:

- Numero di celle
- Numero di nutrienti
- Massimo numero di microorganismi iniziali per ogni specie batterica
- Massimo numero di nutrienti iniziali per ogni tipo
- Numero di iterazioni
- Origine seed
- Interazione di ogni specie coi nutrienti

Per poter implementare la GUI è stato necessario utilizzare alcune librerie che permettano l'elaborazione e la visualizzazione dei dati; è stato deciso di utilizzare *Pandas* e *Plotly*. La prima di queste è una libreria usata per l'elaborazione, l'esplorazione e la manipolazione dei dati. La seconda libreria permette di creare, manipolare e visualizzare una vasta gamma di grafici, perfettamente compatibile col framework utilizzato (*Dash*).

Dash è una libreria di Python utilizzata per lo sviluppo di applicazioni web ed è particolarmente efficiente per lo sviluppo di applicazioni d'analisi e visualizzazione dati. Quello che va a determinare l'aspetto visivo dell'app è il layout.

Lo stile della visualizzazione dei dati ottenuti dalla simulazione è rimasto lo stesso rispetto alla prima versione implementata, ossia la rappresentazione mediante grafici che riportano la quantità dei batteri e nutrienti presenti. Sono state però apportate delle modifiche che possono permettere la visualizzazione dei dati normalizzati oppure considerare solo una determinata porzione. É stato inoltre introdotto anche un pulsante la cui funzione è quella di estrarre i dati in una tabella excel.

Come si può vedere in Figura 1.5, la parte superiore della GUI è dedicata alla scelta dei parametri, mentre quella inferiore è relativa alla visualizzazione dei dati prodotti dalla simulazione. Di seguito verranno elencate le funzioni dei vari riquadri:

- *Seeds*, in questa sezione è possibile manipolare due valori. Il primo relativo alla libreria *random* ed è caratteristico dei vettori del metabolismo e della tossicità, della numerosità di batteri per specie presenti nella cella iniziale ed anche la quantità di nutrienti. Il secondo è relativo alla libreria *numpy* e definisce il tasso di crescita massimo di ogni specie. Avere dei valori di seed noti permette di replicare una simulazione che è già stata effettuata in passato, rendendo così il simulatore randomico ma riproducibile, permettendo anche di effettuare delle simulazioni significative, potendole scegliere da un menù a tendina. Cliccando i pulsanti *RUN* e *DOWNLOAD DATA* è possibile eseguire la simulazione ed esportare i dati.

- *Simulation settings*, in questa sezione è possibile modificare i parametri descritti poco sopra, ad eccezione del numero di specie dato che questa funzione è al momento disabilitata. É possibile modificare questi parametri grazie a caselle di input. Il pulsante *SET DEFAULT VALUES* permette di ripristinare i valori iniziali.

- *Bacteria's parameters*, in questa sezione è presente una tabella in cui sono de-

scritte le interazioni tra i nutrienti e le varie specie batteriche. Nella colonna *Interaction* è presente il metabolismo del batterio, nella colonna *Toxicity* sono presenti le informazioni su quale metabolita risulta tossico per una determinata specie. Ogni riquadro della tabella risulta modificabile come una casella di input.

- *Metabolites Variation Over Time*, è una sezione in cui è presente un grafico raffigurante la variazione dei metaboliti nel tempo. Il numero di grafici presenti equivale al numero di celle considerate nella simulazione. É presente un menù a tendina che permette di visualizzare solamente un determinato sottoinsieme di celle, quindi di grafici.

- *Bacteria Distribution*, è una sezione in cui è presente un grafico raffigurante l'andamento dei batteri nelle varie celle in funzione del numero di iterazione passate. Un menù a tendina permette di selezionare solo alcune delle specie presenti. É possibile interagire col grafico in vari modi: scegliendo come si vogliono visualizzare i risultati (frequenze assolute, normalizzate rispetto alla cella o rispetto al sistema), come cambiare la tipologia di grafico (stackplot o grafico a barre) e osservare come sviluppano le specie in funzione del tempo grazie ad uno *slider*.

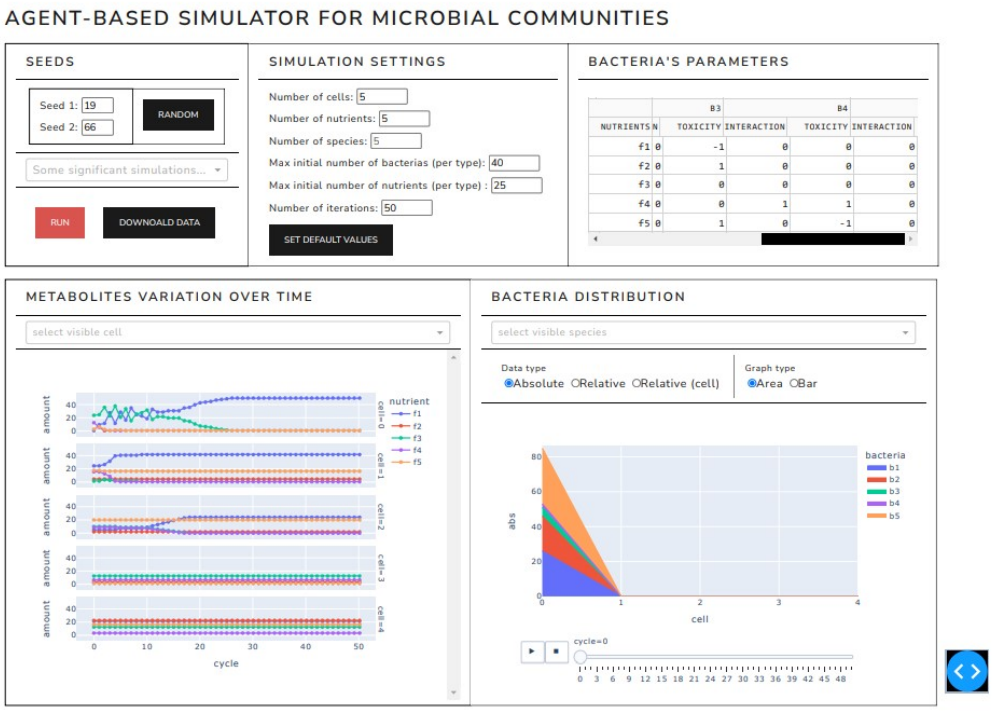


Figura 1.5: Interfaccia grafica di BactLife

Quanto appena descritto è stato implementato tramite delle *Callback Functions*, ossia delle funzioni che vengono passate come argomenti di altre funzioni. Quelle che sono state sviluppate nella tesi di S. Rebecca [3] sono:

- *random_simulation*, ha la funzione di annullare la selezione manuale dei seed quando vengono inizializzati in modo casuale;
- *set_parameters*, ha il compito di creare la tabella dei parametri con i dati degli oggetti *Bacts* ricevuti in input. Questa tabella viene aggiornata ogni volta che vengono modificati i valori dei seed o la quantità di batteri o nutrienti. Ha anche il compito di cambiare i valori del seed nel caso in cui venga cliccato il pulsante *RANDOM* oppure si scelga una simulazione nota;
- *set_default_values*, imposta i valori standard nella sezione *Simulation Settings*, quando viene schiacciato il bottone *SET DEFAULT VALUES*;
- *main*, all'interno di questa funzione vengono creati gli oggetti e vengono inoltre elaborati i dati della simulazione;
- *options_update*, nel caso in cui venissero modificate la quantità di celle e di nutrienti tramite input, questa funzione modifica le opzioni dei menù a tendina;
- *graph_nut*, restituisce in output il grafico che descrive l'andamento dei nutrienti;
- *graph_bact*, restituisce in output il grafico che descrive l'andamento dei batteri nel tempo.

Capitolo 2

Scopo della tesi

Lo scopo di questo elaborato è studiare il problema della reperibilità ed implementabilità dei dati relativi ai terreni di coltura e ai metabolismi dei batteri nei simulatori di comunità microbiche. L'esempio di simulatore che viene preso in considerazione è BactLife in cui, come è stato illustrato nel capitolo 1, i dati implementati sono casuali, non permettendo così di effettuare delle simulazioni scientificamente rilevanti.

Per poter risolvere il problema introdotto è stato studiato lo stato dell'arte dei database al cui interno sono presenti informazioni dei metabolismi di batteri, quali:

- BacDive
- AGORA
- AGREDA

Per implementare dati riguardanti i terreni di coltura così da avere un ambiente di crescita realistico è stato studiato MediaDive.

Una volta studiato il tipo di dato si è pensato una soluzione implementativa affinché si potessero importare nei simulatori i dati in questione, dando alle simulazioni maggiore valenza scientifica.

Capitolo 3

Reperibilità dei dati relativi ai terreni di coltura e ai nutrienti

Le comunità microbiche per potersi evolvere hanno la necessità di trovarsi in un ambiente i cui nutrienti siano non tossici e siano in grado di attivare i loro metabolismi. Dato che si considerano una pluralità di specie, che provano a svilupparsi nella stessa cella, i metaboliti prodotti possono influenzare la crescita delle altre specie, sia positivamente che negativamente.

Ad ora nel simulatore BactLife i nutrienti presenti sono generati casualmente e sono semplici astrazioni di metaboliti generici senza una vera controparte biologica; questo comporta avere una simulazione non veritiera di come i batteri popolino le varie celle all'aumentare delle iterazioni. Per poter provare a risolvere questa problematica e poter effettuare delle simulazioni più realistiche è stato studiato un database chiamato MediaDive, contente le informazioni su più di 3.200 terreni [7].

3.1 Struttura di MediaDive

In questo database i terreni sono formati da delle *solutions*, ognuno ne ha almeno una, ma ne esistono alcuni che sono formati da più di 10. Queste componenti possono essere realizzate tramite l'utilizzo di *solutions* e anche con l'introduzione degli *ingredients*. Spesso è riportata la descrizione del processo da seguire per preparare la soluzione o il terreno preso in considerazione.

La *main solution*, definita da MediaDive come punto di partenza per la preparazione di ogni *media*, è singolare per ogni terreno. Questa *solution* univocamente definita per

ogni terreno presente nel database ha tre particolari caratteristiche:

- riportare tutti gli *ingredients* e *solutions* necessari alla preparazione del terreno. Sono indicate le quantità necessarie delle varie componenti così da facilitare la reperibilità delle informazioni necessarie per la produzione;
- riportare i vari aggiustamenti per i casi specifici, dato che le soluzioni generali potrebbero essere utilizzate in molteplici casi mentre la *main solution* è unica;
- indirizzare alle altre *solutions* direttamente dalla main.

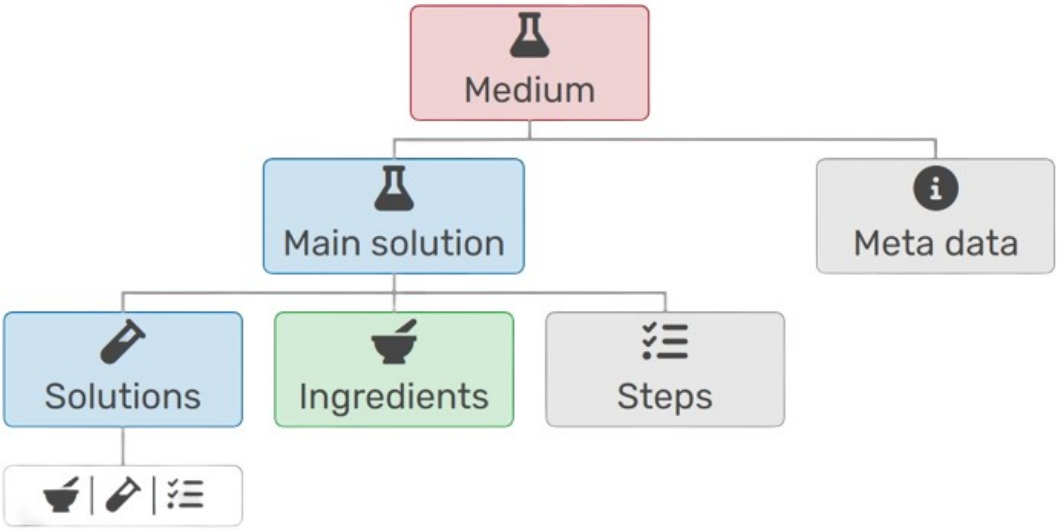


Figura 3.1: Struttura di un terreno di coltura in MediaDive

Come si può vedere nello schema riportato in figura 3.1, ogni terreno ha una *main solution* e una lista di informazioni, come ad esempio i microorganismi che possono crescervi all'interno, la fonte delle informazioni e il pH finale. La *main solution* a sua volta si divide nelle *solution*, negli *ingredients* che la compongono e negli step necessari per poterla produrre in laboratorio. Ogni soluzione a sua volta riporta gli ingredienti e le soluzione che servono per la sua elaborazione.

In figura 3.2 è schematizzata la struttura del database, formato da 5 tabelle principali denominate *"core tables"*, la cui caratteristica principale è la connessione con le 13 tabelle secondarie. Le *core tables* sono:

- media
- strain
- modifications
- ingredients

- solutions

Le linee tra le varie tabelle sono la rappresentazione grafica delle connessioni tra la chiave principale (riconoscibile dato che è affiancata da un simbolo di una chiave) e le chiavi esterne di un'altra tabella (riconoscibili essendo evidenziate in grassetto).

La chiave principale è una colonna, o una combinazione di colonne, che risulta essere unica per ogni riga, mentre la chiave esterna è una colonna, o una combinazione di colonne, che è chiave principale per un'altra tabella.

É riportata anche il tipo di relazione che può essere uno a uno o uno a molti; la chiave primaria è sempre affiancata da un 1, mentre quelle esterne possono essere affiancate da un 1 o da un *. Nel caso in cui ci siano da entrambe le parti un 1 si ha una relazione uno a uno, se nella chiave primaria si ha un 1 e in quella esterna si ha un * si è nella casistica uno a molti. La uno a uno implica che per ogni chiave primaria esiste uno e un solo record (riga) nella tabella associata. La uno a molti rappresenta una relazione in cui ogni chiave primaria può essere riferita ad uno o più record nella tabella associata.

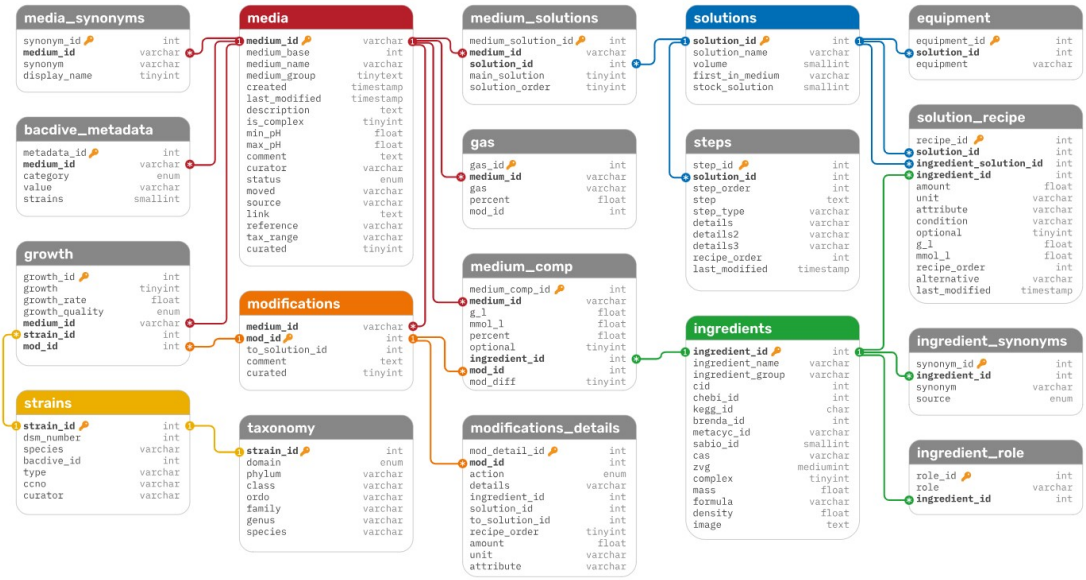


Figura 3.2: Struttura del database MediaDive, fonte `https://mediadive.dsmz.de/docs/website`

3.2 Tipologia dei dati in MediaDive

I numerosi terreni e ingredienti presenti in MediaDive sono molto elevati rispetto a quanti vengono solitamente utilizzati nei laboratori. Ciò fa pensare che molti di questi terreni siano estremamente specifici e siano utilizzati solo in modo altamente selettivo

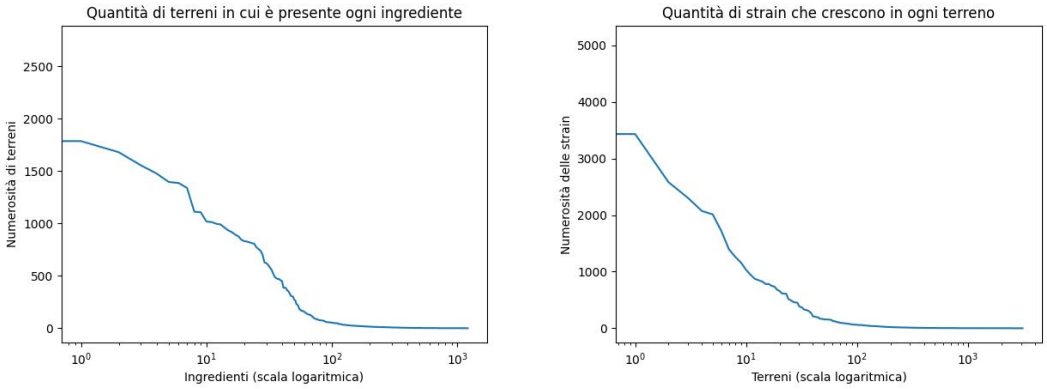
per alcune specie o ceppi batterici, facendo dedurre che molti degli ingredienti siano usati solo per condizioni sperimentali rare, se non uniche.

Per verificare le ipotesi appena esposte sono stati costruiti due grafici: il primo rappresenta la quantità di terreni in cui ogni singolo ingrediente è presente (figura 3.3a), il secondo rappresenta la quantità di ceppi batterici che possono crescere in ogni terreno (figura 3.3b). I grafici sono stati raffigurati in scala logaritmica per evidenziare la drastica differenza tra terreni e nutrienti sovra e sotto rappresentati.

Nella figura 3.3a si può notare che già dopo i primi 60 ingredienti, il grafico tende ad avere un plateau prossimo a zero. Questo è indice del fatto che la quantità di nutrienti che sono necessari per la crescita di molti microorganismi è la minima parte rispetto a tutte le informazioni presenti nel database. Il totale di ingredienti presenti in Mediadive è di 1.211. Le informazioni ricavate permettono di dedurre, che una grossa quantità di dati dei nutrienti siano relativi a terreni specifici di alcuni *strain*. Tale livello di dettaglio è stato ritenuto eccessivo e non fondamentale per lo sviluppo di un simulatore volto a descrivere le evoluzioni di molteplici specie contemporaneamente.

Nella figura 3.3b è possibile visualizzare il numero di diverse specie batteriche in grado di poter crescere per terreno. Si può constatare che il trend è molto simile alla figura adiacente. In questo caso è molto marcato il divario tra i terreni più selettivi e quelli che permettono lo sviluppo di una quantità di specie elevata. Nei primi 10 *media* si può notare un forte incremento della selettività, dato che il numero di *strain* diminuisce da circa 3.500 a meno di 2.500. Dopo 40 terreni è visibile l'inizio del plateau tendente allo zero. Questo permette di constatare che della elevata quantità di *media* presenti (3.294), molti sono di scarso interesse per il simulatore. Se prendessimo in considerazione terreni troppo selettivi infatti, non sarebbe possibile studiare la crescita di molteplici specie contemporaneamente.

Raccolta delle informazioni. I dati di questi grafici sono stati presi dal sito di MediaDive [7]. Inizialmente sono stati scaricati due file .json contenenti alcune informazioni degli ingredienti e dei terreni; è possibile fare ciò direttamente dalle pagine relative sul sito. Di seguito sono stati scritti dei codici in Python; grazie agli id ,presenti nei file .json, è stato possibile accedere alle pagine web specifiche per ogni terreno/ingrediente. Da qui sono stati ricavati i dati relativi alla quantità di terreni in cui è presente ogni ingrediente e la quantità di ceppi batterici che crescono in ogni terreno.



(a) Grafico ingredienti terreni

(b) Grafico terreni strain

Figura 3.3: Grafici MediaDive

3.3 Implementazione di MediaDive in Python

Per poter accedere alle informazioni presenti sul sito internet sono utilizzabili le API (Application Programming Interface) di MediaDive. Nel *Manual* del database sono presenti le istruzioni e degli esempi da poter seguire così da poter capire come utilizzare correttamente le API.

Nel manuale di utilizzo del database sono illustrati molti link che permettono di reperire diverse informazioni ed è mostrata anche la struttura di come si presentano i dati dal link in questione. Ad esempio per ricavare il numero di terreni in cui è presente un ingrediente è stato utilizzato il seguente indirizzo:

https://mediadive.dsmz.de/rest/ingredient/

Per poter aprire e leggere il contenuto di questi link è stata utilizzata una libreria di Python, *urllib*.

Dopo l'ultimo *slash*, inserendo l'id di un ingrediente, viene visualizzato ciò che è riportato sul sito per esso. In figura 3.4 è rappresentato l'esempio del glucosio. La prima informazione che è stata ricavata è il numero di media in cui esso è presente per poter effettuare il grafico in figura 3.3a. Quindi sono stati anche ricavati i sinonimi dei vari ingredienti, l'utilizzo di questi sarà illustrato nella sezione 5.2. Nell'appendice A.1 è riportato il codice per poter ottenere un file .json, in cui è presente un dizionario che ha come chiavi gli ingredienti e come valori un intero rappresentate la quantità di terreni di cui fa parte. Successivamente, il dizionario è stato ordinato in ordine decrescente così che il grafico, in figura 3.3a, fosse interpretabile correttamente.

Dai link presenti nel manuale di utilizzo di MediaDive si ottiene un dizionario. Per

poter ricavare i *media* che hanno tra le proprie componenti l'ingrediente in questione (prendiamo in considerazione ad esempio il glucosio), si deve accedere alla chiave *'data'*, che ha al suo interno a sua volta un dizionario. Esso contiene tutto ciò che è presente in figura 3.4. Sarà poi necessario accedere alla chiave *'media'* in cui è riportata una lista con tutti gli id dei *media* che hanno una composizione che comprenda anche l'ingrediente considerato. Per ricavare i sinonimi invece una volta che si è fatto l'accesso alla chiave *'data'* è necessario far riferimento alla chiave *'synonyms'*, da cui si estrae una lista con tutti i sinonimi dell'ingrediente. Questo script è riportato in A.2.


```
{
    "id": 5,
    "name": "Glucose",
    "ChEBI": 17234,
    "CAS-RN": "50-99-7",
    "KEGG-Compound": "C00293",
    "BRENDA-Ligand": null,
    "MetaCyc": null,
    "PubChem": 962,
    "ZVG": 19010,
    "complex_compound": 0,
    "mass": 180.16,
    "formula": "C6H12O6",
    "density": 1.544,
    "synonyms": [
        "(+)-Glucose",
        "D-Glucose",
        "Glc",
        "gluco-hexose",
        "Glucopyranose",
        ...
    ],
    "roles": [],
    "media": [
        3,
        7,
        8,
        ...
    ]
}
```


Figura 3.4: Informazioni ottenibili dalle API di MediaDive per gli ingredienti

Per riuscire a scaricare i dati per poter fare il plot del grafico in figura 3.3b è stato necessario ricavare le informazioni dal seguente url:

https://mediadive.dsmz.de/rest/medium-strains/

Qui è possibile trovare, per ogni terreno, quali specie batteriche sono in grado di potersi evolvere in esso. Un esempio di terreno, il *NUTRIENT AGAR*, si può vedere in figura 3.5. Sono riportate le prime due *strain* che possono crescere su questo media. Nel primo dizionario sono presenti 3 chiavi:

- *status*
- *count*
- *data*

Una volta fatto accesso a *'data'* è possibile contare i dizionari che esso ha al suo interno per sapere quante *strain* possono svilupparsi in questo terreno. Questa lista è stata ordinata in modo decrescente così da rendere più chiaro il grafico in figura 3.3b. Lo scopo del codice riportato nell'appendice A.3 è quello di creare un file .json contenente un dizionario le cui chiavi sono gli id dei *media* e i valori un intero che rappresenta le *strain* che crescono in esso.


```
{
    "status": 200,
    "count": 2266,
    "data": [
        {
            "id": 1,
            "species": "Weizmannia coagulans",
            "ccno": "DSM 1",
            "growth": 1,
            "bacdive_id": 654,
            "domain": "B"
        },
        {
            "id": 2,
            "species": "Paenibacillus macquariensis subsp. macquariensis",
            "ccno": "DSM 2",
            "growth": 1,
            "bacdive_id": 11477,
            "domain": "B"
```


Figura 3.5: Informazioni ottenibili dalle API di MediaDive per i terreni

Capitolo 4

Reperibilità dei dati relativi a batteri e ai metabolismi

L'energia necessaria all'accrescimento dei microorganismi è ottenuta dall'ossidazione dei nutrienti; questo avviene grazie ad una vasta gamma di reazioni biochimiche che essi possono compiere. Quando avvengono queste reazioni, oltre alla produzione di energia, vengono anche generati dei prodotti (costitutivi o di scarto). Ci sono una vasta gamma di vie metaboliche che permettono, solitamente, partendo dall'ossidazione del glucosio, la produzione di energia. Nei simulatori di comunità microbiche è indispensabile conoscere sia i nutrienti irrinunciabili, per permettere l'evoluzione delle varie specie, che i metaboliti prodotti a seguito delle vie metaboliche attivate; questi saranno presenti nell'ambiente e andranno ad influire nella crescita delle specie presenti.

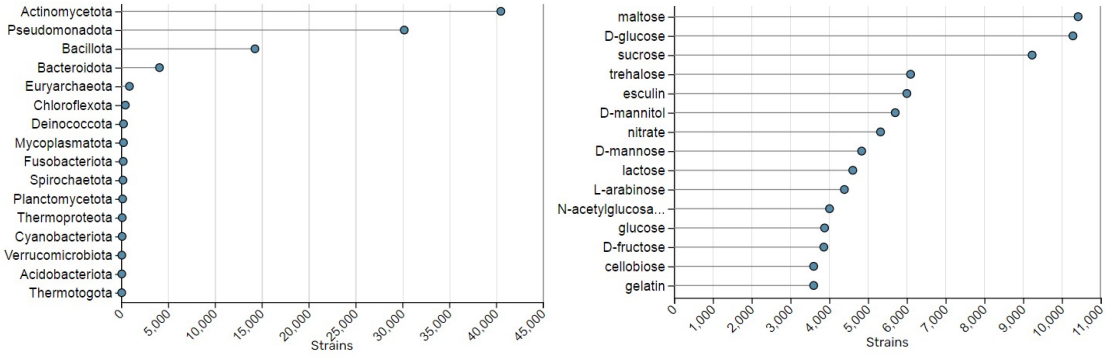
Nel simulatore BactLife ad ora sono assenti dati biologicamente validi dei metabolismi, non permettendo di effettuare delle simulazioni con immediata applicabilità sperimentale. Per provare a ricavare dati che siano realistici, in modo da avere delle simulazioni con parametri relativi ai metabolismi dei batteri quantitativamente plausibili, sono stati studiati una serie di database quali: AGORA, AGREDA e BacDive [6, 8, 9].

I tre database considerati hanno strutture molto diverse, BacDive è molto simile a MediaDive, sia per come è strutturato sia per la strutturazione dei dati contenuti, essendo entrambi sviluppati e curati da DSMZ, collezione tedesca di microrganismi e colture cellulari. AGORA ed AGREDA hanno invece una costituzione simile tra loro ed estremamente differente rispetto a quella di BacDive, progettati per lavorare in un framework di *flux balance analysis* (FBA) e reti metaboliche.

4.1 BacDive

4.1.1 Struttura del database

Nella homepage del sito BacDive è possibile trovare dei grafici che rappresentano alcuni dati reperibili. Un esempio sono il grafico dei *Phylum* più presenti o il grafico dei metaboliti maggiormente utilizzati, rappresentati in figura 5.1, che aiutano a capire la tipologia di informazioni reperibili in questo database.



(a) Grafico *Phylum*

(b) Grafico metaboliti prodotti

Figura 4.1: Grafici homepage BacDive

In BacDive sono presenti 93.524 *strain* totali, di cui 19.313 sono *type strain*. I *type strain* rappresentano un punto di partenza per l'identificazione di nuovi *strain*, talvolta sono i primi *strain* di una specie ad essere identificati, di cui sono noti il fenotipo ed altre caratteristiche fisiologiche [16].

Una volta effettuato l'accesso al sito è possibile cercare il ceppo batterico preso in considerazione, banalmente digitando il nome su una barra di ricerca, oppure andando nella sezione *advanced searches* applicando dei filtri su *classe*, *phylum*, etc... Una volta selezionato un ceppo batterico all'interno della pagina, si possono trovare le caratteristiche morfologiche ed i terreni di coltura dove può svilupparsi, con la relativa referenza a MediaDive. Nella pagina ci sono molte altre caratteristiche con informazioni riguardanti il ceppo batterico. Per quanto riguarda lo studio analizzato in questo elaborato, risulta di fondamentale importanza ciò che è riportato nella sezione *Phisiology and metabolism*, relativa ai metaboliti utilizzati e prodotti dal ceppo batterico. Nella figura 4.2 sono riportati i dati dei metaboliti utilizzati e prodotti dal batterio preso in esame, *Pseudomonas aeruginosa* [17]. Come è possibile notare non sono riportati i rapporti stechiometrici, che indicano la quantità dei metaboliti usati o prodotti, è solo possibile ottenere una lista di nutrienti e degli scarti metabolici che sono caratteristici per un ceppo batterico.

Metabolite utilization

	Metabolite	Utilization activity	Kind of utilization tested	
[Ref.: #22939]	acetate ChEBI	+	growth	
[Ref.: #68369]	adipate ChEBI	+	assimilation	* from API 20NE
[Ref.: #22939]	alpha-D-glucose ChEBI	+	growth	
[Ref.: #22939]	alpha-lactose ChEBI	-	growth	
[Ref.: #68369]	arginine ChEBI	+	hydrolysis	* from API 20NE
[Ref.: #22938]	beta-alanine ChEBI	+	growth	
[Ref.: #22939]	beta-phenylethylamine ChEBI	-	growth	
[Ref.: #22938]	casamino acids	+	growth	
[Ref.: #22939]	casein	+/-	hydrolysis	
[Ref.: #22939]	citrate ChEBI	+	growth	

Only first 10 entries are displayed. Click here to see all.

(a) Alcuni metaboliti utilizzati da *Pseudomonas aeruginosa* [17]

Metabolite production

	Metabolite	
[Ref.: #22938]	acetoin ChEBI	
[Ref.: #22939]	fluorescein ChEBI	
[Ref.: #22938]	hydrogen sulfide ChEBI	
[Ref.: #22938]	indole ChEBI	
[Ref.: #22938]	pyocyanine ChEBI	
[Ref.: #22939]	pyocyanine ChEBI	
[Ref.: #68369]	indole ChEBI	* from API 20NE

(b) Metaboliti prodotti da *Pseudomonas aeruginosa* [17]

Figura 4.2: Metaboliti utilizzati e prodotti da *Pseudomonas aeruginosa*

4.1.2 Analytical Profile Index (API)

Per poter capire quali siano le reazioni che un microorganismo compie, BacDive fa riferimento al test Analytical Profile Index (API ®) per esplicitare i metaboliti utilizzati e prodotti. API è composto da una serie di test ognuno dei quali rappresenta una differente reazione fisiologica, che potrebbe essere svolta dal microorganismo in considerazione. Questa reazione può servire per capire l'attività di un enzima e/o per vedere se un metabolita viene utilizzato. Il test viene svolto su delle piastre andando a verificare se avviene un cambiamento del colore o della torbidità; i risultati vengono riportati con dei segni che rappresentano:

- + positivo (enzima attivato, metabolita utilizzato)
- - negativo (enzima non attivato, metabolita non utilizzato)
- +/- debole

4.1.3 Application Programming Interface (API)

Per poter utilizzare i dati presenti su BacDive con linguaggio Python si possono usare le API (Application Programming Interface) del database stesso. Come prima cosa si

deve installare e importare il pacchetto. É possibile effettuare l'installazione da riga di comando utilizzando la seguente istruzione:

```
pip install bacdive
```

Una volta importato il pacchetto nel file si deve effettuare il login con le proprie credenziali del sito (email e password). Per poter utilizzare le API è necessario creare un account sul sito del database. Nelle prime righe di codice presente in appendice B.1 si può vedere come effettuare il login alle API, nelle righe successive sono presenti delle istruzioni illustranti l'utilizzo di alcuni metodi.

In queste API sono presenti due metodi, *search* e *retrive*.

Search serve per effettuare delle ricerche grazie a dei parametri:

- ID di BacDive del microorganismo (è possibile anche cercare più di un id alla volta) [id]
- Culture collection number; ossia il numero identificativo del batterio in alcuni depositi, come può essere ad esempio ATCC (American Type Culture Collection) [culturecolno]
- 16S sequence access; un metodo per l'identificazione di batteri grazie al sequenziamento dell'RNA ribosomale 16S che è specifico per ogni microorganismo [16S]
- Genome accessions, numeri identificativi di determinate sequenze di DNA o proteine [genome]
- Infornazioni relative alla tassonomia [taxonomy]

Il metodo *search* ricava tutti gli id, nell'intera banca dati, che soddisfino la caratteristica richiesta.

Il metodo *retrieve* si utilizza dopo *search* e serve per ottenere dati specifici dagli id presi in considerazione. Utilizzando *retrieve* si ottiene un dizionario di tutte le informazioni presenti nel database per i microorganismi di interesse. Si possono applicare dei filtri a questo metodo così da non dover visualizzare tutto ciò che è presente nel database, ma solo quello che è necessario. Per poterlo fare basta richiamare la chiave del dizionario che come valori ha ciò che si sta cercando. É possibile applicare più filtri contemporaneamente chiamando più chiavi insieme.

4.2 AGORA

AGORA [8] è un repository di una banca di dati chiamata Virtual Metabolic Human (VMH) [18]. Su VMH sono presenti dati del *human gut metabolism* che metteno in connessione queste informazioni con alcune malattie. Sono riportate le informazioni di 818 ceppi batterici differenti. Per ognuno di essi sono presenti una serie di caratteristiche, come ad esempio le reazioni che avvengono al suo interno ed i metaboliti che sono utilizzati e prodotti, tuttavia non è presente una distribuzione esplicita se un metabolita ricada in un caso o nell'altro, in quanto in reazioni diverse può assumere ruoli diversi. Per poter capire la stechiometria di ogni nutriente sarebbe , ad esempio, necessario analizzare tutte le reazioni così da comprendere quando sono dei reagenti e quando sono dei prodotti, andando così a stimare se al netto di tutte le reazioni, il nutriente è mediamente più utilizzato o più prodotto. Le numerosità di metaboliti e di reazioni presenti per ogni microorganismo sono molto elevate, ad esempio per il batterio *Acinetobacter baumanii* sono riportati 1.593 reazioni e 1.181 metaboliti.

Per analizzare questa elevata quantità di dati viene utilizzato un toolbox di MATLAB, COnstraint-Based Reconstruction and Analysis Toolbox (COBRA toolbox) [19], illustrato nella sezione 4.4.

4.2.1 Modelli dei microorganismi

La differenza sostanziale di questo database rispetto a BacDive e MediaDive è essere implementato in MATLAB anzichè in Python. AGORA è costituito da 818 modelli, formati da file .mat; questi sono scaricabili dal repository di AGREDA. Quando vengono importati in MATLAB, sono caricati come delle *struct*, al cui interno si possono trovare svariati vettori e matrici.

Il dato di maggiore rilevanza all'interno del modello è la matrice sparsa *S*, che è la matrice stechiometrica associata alle reazioni metaboliche del batterio in questione. In questa matrice, le righe rappresentano i metaboliti utilizzati o prodotti, mentre le colonne rappresentano le reazioni; al suo interno sono presenti i valori:

- +1 indica che il metabolita è stato utilizzato
- -1 indica che il metabolita è stato prodotto

All'interno della matrice stechiometrica *S* è presente una reazione associata alla formazione di biomassa, rappresentante un bilancio dei metaboliti che vengono ingeriti e quali vengono fatti fuoriuscire dal batterio, permettendo così la produzione di bio-

massa. Per riuscire a individuare questa particolare reazione presente nel modello è necessario utilizzare una funzione del *COBRA* toolbox di MATLAB, chiamata *findSExRxnInd*. Tra gli output di questa funzione applicabile alla *struct* del batterio si ha un vettore (il cui nome è *biomassBool*), la cui lunghezza è pari al numero di metaboliti e contenente tanti zeri quanti sono i metaboliti che non prendono parte alla formazione di biomassa. I restanti hanno un valore che è positivo nel caso siano utilizzati e negativo nel caso in cui siano prodotti di scarto; questo rappresenta il coefficiente stechiometrico per la produzione di biomassa.

4.3 AGREDA

AGREDA [9] è un repository, più recente rispetto ad AGORA, contenente modelli metabolici dell' *human gut microbiome (HGM)*. Sono state inoltre incluse informazioni relative a diverse vie metaboliche associate a molteplici diete. Nello specifico in AGREDA sono state implementate 179 vie metaboliche di composti fenolici, aventi una grande rilevanza nella salute e nella dieta umana. Come per AGORA, anche i modelli presenti in questa banca dati sono utilizzabili in linguaggio MATLAB. La quantità di microorganismi considerati è la stessa di AGORA, ossia 818.

4.3.1 Modelli dei microorganismi

La caratteristica principale di AGREDA è nella strutturazione dell'informazione che non ha più un file .mat per ogni batterio preso in considerazione, bensì esiste un unico file dove è possibile reperire tutti i dati di ogni ceppo batterico; questo si può trovare nel repository del database. Una volta implementato il file in questione su MATLAB viene caricata una *struct*, nominata *model*. Il dato principale all'interno di questo modello è la matrice stechiometrica *S*; essendoci una sola *struct*, non sono contenute 818 matrici stechiometriche, ma è presente solamente una matrice *S* con 2.602 righe, rappresentati i metaboliti, e 5.944 colonne, rappresentati le reazioni. Questa particolare matrice è stata creata grazie ad un algoritmo di *gap filling* [9] spiegato nella successiva sezione 4.3.2.

Per poter individuare quali siano le reazioni che un determinato batterio compie, essendoci un'unica matrice stechiometrica, nel modello è presente una matrice sparsa, chiamata *rxnTaxMat* composta da 818 colonne e 5944 righe. Ogni colonna riporta le reazioni che il microorganismo di interesse compie. *rxnTaxMat* è una matrice di 1 e 0, in cui 1 indica che la reazione avviene mentre 0 che non avviene. Le reazioni sono

nello stesso ordine in cui sono riportate nella matrice *S* e il loro nome è anche esplicitato nel vettore *rxns*. Per vedere in che posizione sia un batterio in *rxnTaxMat* si può consultare un altro vettore, *modelID*, in cui son riportati i nomi in ordine di tutti le strain batteriche.

4.3.2 Algoritmo gap filling

L'algoritmo di *gap filling* [9] è utilizzato per la creazione della matrice stechiometrica *S* di AGREDA, che rispetto ai dati presenti su AGORA, è implementata da un maggior numero di composti presenti nelle diete umane e le corrispettive vie di degradazione metabolica. Questi nutrienti sono stati ricavati dal sito i-Diet (`http://www.idiet.es`).

Il primo passo è creare un subset contenente le reazioni, che prende il nome di *core*, andando man mano ad inserire tutte le reazioni per ottenere un dataset universale. Alla base dell'algoritmo di *gap filling* c'è una funzione del *COBRA* toolbox, *FastCoreWeighted*, che prendendo in input un modello e il *core* delle reazioni, restituisce in output un nuovo modello più piccolo possibile che soddisfi i vincoli e inserisca le reazioni presenti nel *core*.

Il *gap filling* è un algoritmo iterativo, la prima iterazione consiste nell'inserire tutte le reazioni presenti in AGORA, così da avere tutte le informazioni di base sugli 818 microorganismi. Successivamente sarà necessario introdurre le reazioni relative ai 221 nuovi nutrienti, presi da i-Diet. Prendendo un nutriente alla volta vengono inserite le vie metaboliche in cui esso è coinvolto, aggiungendo al modello anche gli altri metaboliti che prendono parte in queste vie. Una volta che sono terminate le 222 iterazioni, una per inserire i dati di AGORA e 221 per i nutrienti di i-Diet, è stato ottenuto un network con 2.920 metaboliti e 6.277 reazioni. 51 di queste reazioni sono senza alcuna tassonomia assegnata rispetto alle specie di AGORA, di conseguenza sono state eliminate. É stato necessario eliminare altre reazioni che non fossero strettamente correlate con l'HGM. Questo è stato possibile implementarlo con un'altra funzione del *COBRA* toolbox, *fastFVA*. La funzione appena nominata applica un algoritmo il cui modello alla base è la *FVA (Flux Variability Analysis)*, avente lo scopo di determinare il range dei flussi metabolici, ossia lo spostamento di metaboliti nei percorsi metabolici cellulari, tali per cui sia sempre soddisfatta la *FBA (Flux Balance Analysis)* iniziale, ottenendo così dai flussi metabolici le reazioni fondamentali per il metabolismo. La FBA è un metodo matematico che permette di svolgere simulazioni sui metabolismi, in sostanza riesce a cercare tra migliaia di reazioni possibili le vie metaboliche che

mettono in relazione dei metaboliti; inoltre è computazionalmente molto poco dispendiosa [20]. Dopo l'utilizzo della funzione *fastFVA* si è arrivati ad ottenere un *core* con 2.742 metaboliti e 6.122 reazioni. Successivamente sono state svolti ulteriori aggiustamenti ottenendo il *core* finale del *gap filling* con 2.720 metaboliti e 6.088 reazioni. Come si può notare il dataset finale di reazioni e metaboliti non corrisponde a quello ottenuto alla fine del *gap filling*, dato che la matrice stechiometrica *S* ha 2.602 metaboliti e 5.944 reazioni, quindi il modello è stato ulteriormente raffinato. Uno schema rappresentate l'algoritmo appena illustrato è in figura 4.3

4.4 COBRA toolbox

Il COnstraint-Based Reconstruction and Analysis Toolbox (COBRA toolbox) [19] è un toolbox di MATLB ed in questo elaborato è stato utilizzato per poter analizzare dei dati presenti su AGORA ed AGREDA. Trova molteplici utilizzi in biologia e biotecnologie dato che si può utilizzare con qualunque network biochimico.

L'approccio generale di *COBRA* è rappresentare con un modello meccanicistico basato su vincoli, la relazione tra il genotipo e il fenotipo, grazie a modelli matematici e computazionali. Al suo interno sono presenti una varietà di metodi di analisi, due di questi (FVA e FBA) sono stati esposti nella sezione precedente sul *gap filling* 4.3.2.

Per poter utilizzare questo tool innanzitutto è necessaria installarlo ed i comandi per poterla effettuare sono presenti nella repository di *COBRA* (`https://github.com/opencobra/cobratoolbox`). Ogni volta che lo si vuole utilizzare è indispensabile inizializzare il pacchetto col comando:

```
initCobraToolbox()
```

I modelli che il toolbox è in grado di leggere possono essere di diversi formati:

- file con estensione .xml
- fogli Excel .xls
- MATLAB file .mat

É presente una guida che spiega le funzioni presenti, a cosa servono e come vadano implementate in un codice [21].

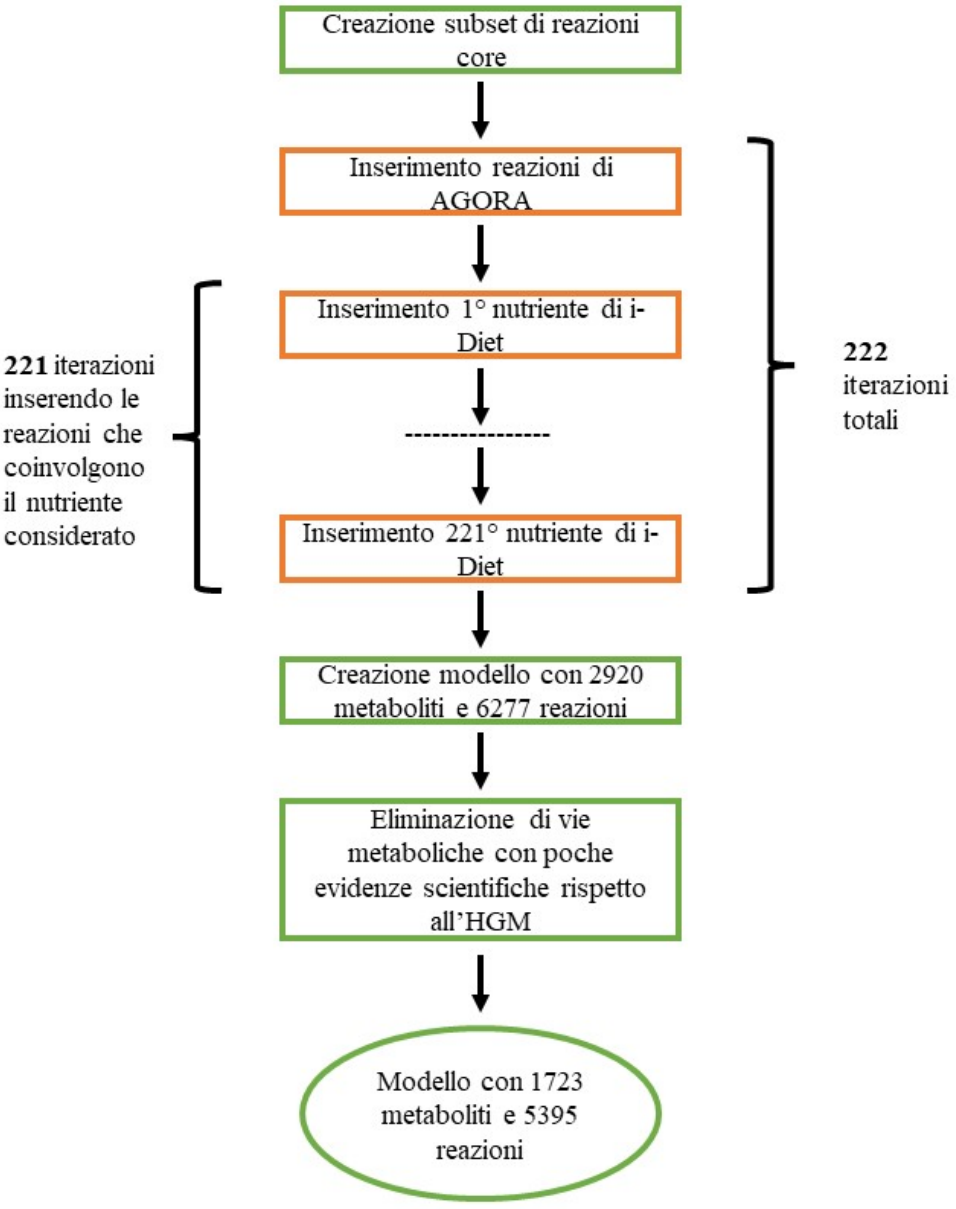


Figura 4.3: Schema rappresentante il funzionamento dell'algoritmo *gap filling*

4.4.1 COBRApy

Esiste un pacchetto di Python, *COBRApy* [22], in cui è stato implementato il *COBRA* toolbox, così da poter sfruttare le sue potenzialità anche su Python e non solo su MATLAB. Con questo pacchetto è possibile effettuare delle analisi come la FBA e la FVA in Python.

In figura 4.4 sono rappresentate le classi di *COBRApy*, tre di esse definiscono il *core* del pacchetto:

- *Model*, rappresenta gli organismi. Si potrebbe pensare come un contenitore di tutte le reazioni e i metaboliti.
- *Reaction*, rappresenta le reazioni del modello;
- *Metabolite and Gene*, rappresentano le biomolecole e i geni.

Le tre classi appena descritte sono tutte connesse tra loro, ad esempio se si prendesse in considerazione un metabolita e si utilizzasse il metodo *get_reaction()* si otterrebbero le reazioni in cui quel metabolita è compreso. Il design del pacchetto basato sugli oggetti permette all'utente di accedere direttamente agli attributi dei vari oggetti, cosa che non è possibile nel toolbox di MATLAB perchè le entità biologiche e i suoi attributi sono contenuti in liste separate.

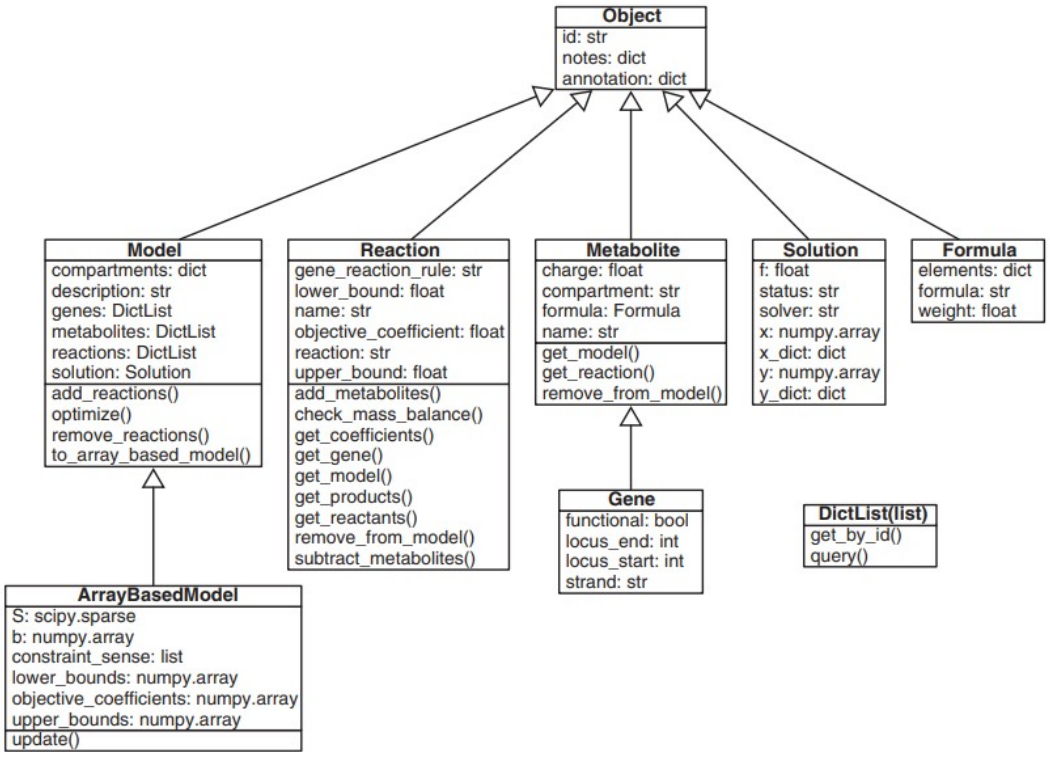


Figura 4.4: Classi COBRApy [22]

Capitolo 5

Aspetti implementativi

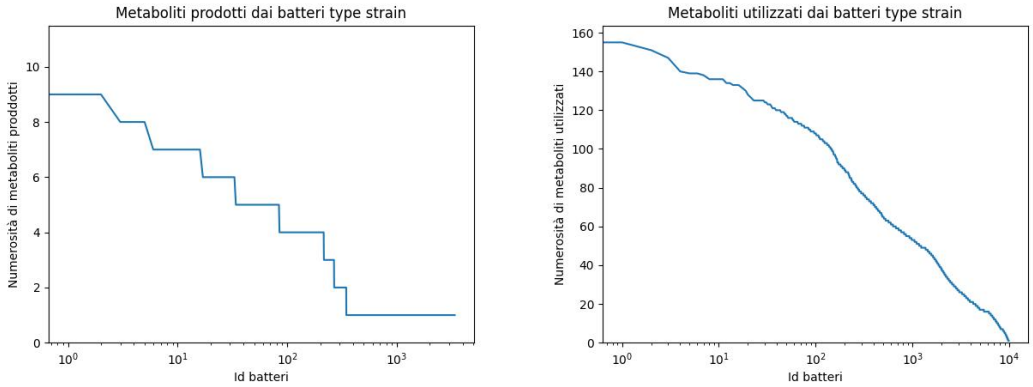
Lo scopo di questo elaborato è riuscire ad analizzare i dati presenti in alcuni database affinché sia possibile ottenere dell'informazione, relativa ai metabolismi ed ai terreni di coltura, utilizzabile in un simulatore di comunità batteriche. Per fare questo è stato necessario fare delle analisi sulle informazioni presenti nei database che sono stati illustrati nei capitoli 3 e 4.

Per riuscire ad estrapolare i dati di interesse è stato necessario scrivere dei codici in Python e in MATLAB, così da poter fare delle analisi su una grande quantità di dati.

5.1 Analisi sulle type strain di BacDive

Data la grande quantità di dati presenti su BacDive (19.313 *type strain*) è stato necessario sogliare i batteri. Utilizzando le API del sito internet sono stati creati due dizionari le cui chiavi sono gli id dei batteri e i valori sono numeri interi. In uno dei due dizionari l'intero, che si ha come valore, rappresenta il numero di metaboliti utilizzati, nell'altro, invece, l'intero rappresenta il numero di metaboliti prodotti. Sono state inoltre create due liste, la prima contenente tutti i metaboliti utilizzati e la seconda quelli prodotti. Tutti i dati ottenuti sono stati salvati in dei file .json. Il tutto è stato implementato nel codice in appendice C.1. In sostanza, nel codice Python sono state utilizzate le API di BacDive, una volta importate e fatto l'accesso, è stato utilizzato il metodo *search* per ogni microorganismo e successivamente il metodo *retrieve* per interagire con la sezione *'Physiology and metabolism'* da cui si ottiene un dizionario. Con la chiave *'metabolite utilization'* si hanno tutti i metaboliti utilizzati da quel batterio, mentre con la chiave *'metabolite production'* si hanno tutti i metaboliti prodotti.

Partendo dai due dizionari creati sono stati elaborarti due grafici, entrambi hanno sull'asse delle ascisse gli id dei batteri (riportati in scala logaritmica) e sulle ordinate si possono trovare i metaboliti prodotti, figura 5.1a, o i metaboliti utilizzati, figura 5.1b. Come si può notare, sui metaboliti prodotti ci sono meno informazioni, sia perchè ci sono poche specie che hanno questi dati, 3.374 *type strain*, sia perchè la numerosità di metaboliti prodotti da ogni specie è bassa. Per i metaboliti utilizzati si ha una mole di informazioni maggiore, di fatto si hanno dati per circa 10.000 *type strain* con una numerosità elevata.



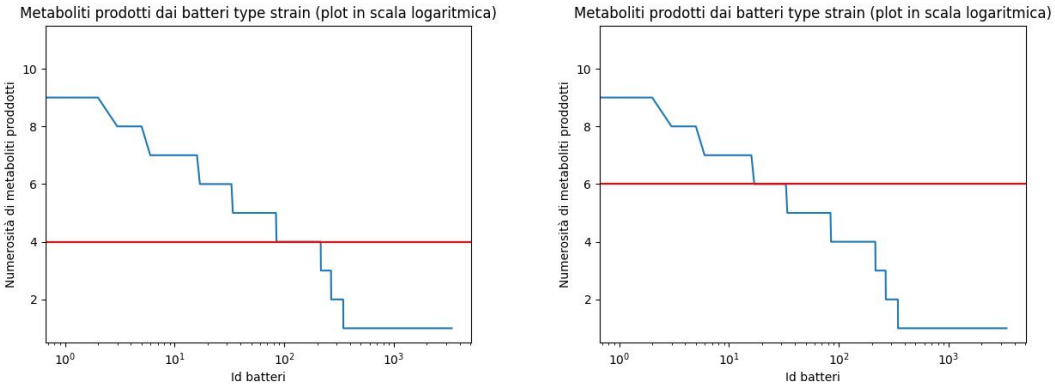
(a) Grafico id-batteri metaboliti prodotti, scala logaritmica

(b) Grafico id-batteri metaboliti utilizzati, scala logaritmica

Figura 5.1: Grafici BacDive

Una prima selezione è stata fatta considerando solo i microorganismi per i quali si avessero informazioni sui metaboliti prodotti, dato che su BacDive sono riportate molte *type strain* senza che ci siano dati sui loro metabolismi. É stata fatta una ulteriore classificazione sulla quantità di metaboliti prodotti considerando i microorganismi aventi più di 4 prodotti, che sono risultati essere 216, e quelli con più di 6 prodotti, che sono risultati essere 34. In figura 5.2 si riproducono i grafici in scala logaritmica dei metaboliti prodotti. Nel grafico in figura 5.2a al di sopra della linea rossa ci sono i batteri con più di 4 metaboliti prodotti e nel grafico in figura 5.2b i valori al di sopra della linea indicano i batteri con più di 6 metaboliti prodotti.

Una volta trovati i microorganismi che hanno dati sufficienti sui metaboliti prodotti è stata fatta un'analisi sui loro metaboliti utilizzati. Considerando i 216 batteri con più di quattro metaboliti prodotti due di questi non hanno dati relativi ai metaboliti utilizzati. Tra i 214 rimanenti si ha una media di 26 metaboliti utilizzati e una varianza di 352. Facendo riferimento alle 34 *type strain*, aventi più di 6 metaboliti prodotti, si può verificare che uno di essi non ha alcun dato sui metaboliti utilizzati, mentre i



(a) Grafico id-batteri metaboliti prodotti con soglia a 4 metaboliti prodotti, scala logaritmica

(b) Grafico id-batteri metaboliti utilizzati con soglia a 6 metaboliti prodotti, scala logaritmica

Figura 5.2: Grafici soglie metaboliti prodotti

restanti 33 hanno una media di 30 metaboliti utilizzati e una varianza di 237. Il codice in cui sono state fatte queste analisi è riportato nell'appendice C.2.

É stato fatto un boxplot, rappresentato in figura 5.3, per visualizzare la distribuzione della quantità di metaboliti utilizzati nei microorganismi aventi più di 4 o 6 metaboliti prodotti. É possibile constatare che nei batteri con soglia pari a 4 siano presenti una quantità elevata di outlier. In entrambi i casi si può notare come gli estremi superiori e inferiori abbiano valori simili e siano circa nello stesso range. Il grafico è stato implementato col codice presente nell'appendice C.2.

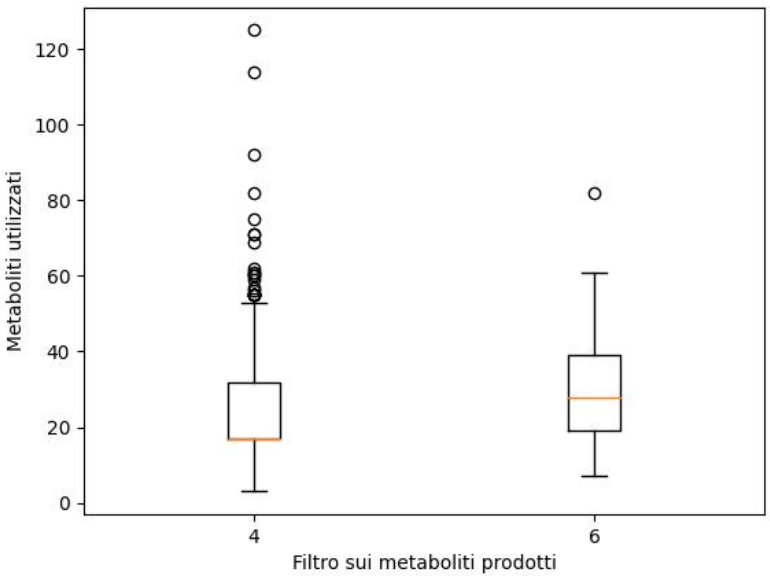


Figura 5.3: Boxplot metaboliti utilizzati dai batteri sogliati sui metaboliti prodotti

Dato che nei simulatori di comunità microbiche i microorganismi si trovano tutti nello stesso ambiente i nutrienti che un batterio può produrre potrebbero influire l'evoluzione di un'altra specie batterica, sia positivamente sia negativamente. Quindi è stato scritto un codice, presente in appendice C.3, affinché venissero prodotti dei dati rispetto all'intersezione tra i metaboliti utilizzati e quelli prodotti. Per le 214 *type strain*, ottenute con soglia pari a 4 metaboliti prodotti e aventi informazioni sui metaboliti utilizzati, si hanno 44 metaboliti che sono presenti sia tra gli utilizzati sia tra i prodotti. Per i 33 batteri, ottenuti con soglia pari a 6 sui metaboliti prodotti e aventi informazioni sui metaboliti utilizzati, l'intersezione è composta da 15 nutrienti.

5.2 Relazione tra MediaDive e BacDive

I database BacDive e MediaDive sono curati dalla *DSMZ*, collezione tedesca di microrganismi e colture cellulari. I due database comunicano tra loro, di fatto è possibile trovare su BacDive informazioni relative a MediaDive e viceversa. In questa sezione saranno illustrate le analisi effettuate per capire se fosse possibile integrare i dati presenti su MediaDive per i terreni di coltura e i dati presenti su BacDive per i metabolismi dei batteri. Data la loro origine comune, si potrebbe pensare che i dati dei metaboliti siano in comune, quindi che i terreni di coltura al loro interno abbiano gli stessi nutrienti dei metabolismi batterici.

Come prima analisi per verificare il tipo di relazione tra i dati di BacDive e MediaDive è stata studiata l'intersezione presente tra i metaboliti di tutti i *type strain* presenti in BacDive e tutti gli ingredienti di MediaDive. Nella tabella 5.1 sono riportate le quantità dei metaboliti prodotti dai *type strain* di BacDive, si può notare che tra i 363 metaboliti prodotti solo 46 di questi sono presenti in MediaDive (rappresentante il 13%), e la numerosità dei metaboliti utilizzati dai microorganismi, in totale sono 906 e di questi 230 sono riportati su MediaDive (rappresentante il 25%). Nel codice che è stato implementato per poter ricavare questi dati, presente nell'appendice C.4, sono stati utilizzati gli ingredienti di MediaDive e i loro sinonimi, inoltre tutte le stringhe che compongono i metaboliti o gli ingredienti sono state riscritte tutte in maiuscolo; questi accorgimenti sono stati presi per poter aver un'analisi il più accurata possibile anche se i dati sono riportati in modo differente nei due database.

L'analisi appena fatta per vedere l'intersezione dei metaboliti di tutte le *type strain* e degli ingredienti di MediaDive è stata replicata considerando solo i microorganismi di BacDive che hanno almeno 6 metaboliti prodotti e informazioni sui metaboliti utiliz-

	Metaboliti prodotti	Metaboliti utilizzati
Metaboliti BacDive	363	906
Metaboliti non presenti in MediaDive	317	676

Tabella 5.1: Intersezione ingredienti MediaDive con metaboliti BacDive

zati, i batteri di interesse sono riportati nella tabella 5.4. Non sono stati considerati quelli con soglia pari a 4 perchè il numero di strain batteriche è elevato per una prima analisi e una possibile implementazione. Nell'effettuare questo studio si è prestata una maggiore attenzione ai metaboliti utilizzati, dato che i nutrienti contenuti nei terreni sono i metaboliti che verranno utilizzati dai batteri presenti in esso. Nella tabella 5.2 che riporta i risultati possiamo notare che per i 197 metaboliti utilizzati dalle 33 *type strain* considerate, 107 di essi sono presenti in MediaDive, ossia il 54%; mentre dei 66 metaboliti prodotti soltanto 18 sono presenti in MediaDive come ingredienti, ossia il 27%.

	Presenti in BacDive	Presenti in BacDive e MediaDive
Metaboliti prodotti dalle 33 *type strain*	66	18
Metaboliti utilizzati dalle 33 *type strain*	197	107

Tabella 5.2: Intersezione ingredienti MediaDive con metaboliti BacDive delle *type strain* con più di 6 metaboliti prodotti

Il dato principale ricavato da questa analisi è che il 54% dei metaboliti utilizzati dalle 33 *type strain*, aventi più di 6 metaboliti prodotti, compongono i terreni di coltura di MediaDive. Questo risultato non permette di avere una quantità di dati sufficientemente elevata per far sì che gli ingredienti di MediaDive possano dare le informazioni necessarie sull'evoluzione delle strain batteriche nei terreni. Quasi metà dei metaboliti utilizzati sono assenti quindi non riusciremmo a capire per molti dei batteri considerati quando effettivamente non sia possibile la crescita o a causa della mancanza di nutrienti o per la mancanza di informazioni del database.

Nella tabella 5.3 sono riassunti tutti i dati calcolati sulle specie che rispettano le due soglie sui metaboliti prodotti e che abbiano anche informazioni relative ai metaboliti utilizzati.

Soglia	#Specie	# Uti	Mean uti	Var uti	# Prod	Mean prod	Var prod	Prod ∩ MediaDive	Uti ∩ MediaDive	Prod ∩ Uti
4	214	362	26.24	351.72	208	4.67	1.21	30	149	44
6	33	196	31.21	237.08	66	6.82	1.30	18	107	15

Tabella 5.3: Tabella riassuntiva con tutti i dati calcolati sulle *type strain* sogliate sui metaboliti prodotti. Uti = utilizzati, Var = varianza, Prod = prodotti, Mean = media, ∩ = intersezione.

In questo studio si è riscontrato che i dati presenti su MediaDive non sono sufficienti rispetto ai metabolismi riportati su BacDive, quindi è stato creata una tabella in un file excel, una parte è raffigurata in figura 5.4, facilmente importabile, in cui sono presenti tutti i metaboliti prodotti e utilizzati dalle 33 *type strain* di interesse, le cui caselle contengono i seguenti valori:

- 0 se il nutriente non è di interesse per la specie;
- +1 se il metabolita viene prodotto;
- -1 se il metabolita viene utilizzato;

	A	B	C	D	E	F	G	H	I
1	**Metabolites**	**140317**	**2601**	**58**	**132584**	**132404**	**2760**	**2853**	**13509**
2	(-)-quinic acid	0	0	0	0	0	0	0	0
3	2,5-didehydro-D-gluconate	0	0	1	0	0	0	0	0
4	2-aminobutyrate	0	0	0	0	0	0	0	0
5	2-dehydro-D-gluconate	0	0	1	0	0	0	0	0
6	2-deoxy-d-ribose	0	0	0	-1	0	0	0	0
7	2-oxoglutarate	0	0	0	0	0	0	0	0
8	3-phenylpropionate	1	0	0	0	0	0	0	0
9	4-hydroxybutyrate	0	0	0	0	0	0	0	0
10	5-dehydro-D-gluconate	0	0	1	0	0	0	0	0
11	D-alanine	0	0	0	0	0	0	0	0
12	D-arabinose	0	0	-1	0	-1	0	0	0
13	D-arabitol	0	0	0	0	-1	0	0	0

Figura 5.4: Sezione tabella metabolismi batteri

Come riportato nella sezione 5.1, l'unione tra i 66 metaboliti prodotti e i 197 metaboliti utilizzati dalle 33 specie batteriche è composta da 15 elementi, conseguentemente la lista senza ripetizioni dei metaboliti che possono essere sia utilizzati che prodotti è formata da 248 composti. Questi metaboliti sono messi in ordine alfabetico e ogni metabolita corrisponde a una riga, mentre una colonna corrisponde a un *type strain*, ottenendo così una tabella con 249 righe (la prima riporta gli id di BacDive delle specie in considerazione) e 34 colonne (la prima riporta tutti i metaboliti presenti e le 33 successive sono i *type strain considerati*). La tabella elaborata può essere implementata in BactLife, aggiornando le variabili illustrate nel paragrafo 1.3:

- la lista *_food* diventerà un sottoinsieme dei 248 nutrienti;
- la lista *_species* definirà quali delle 33 possibili specie batteriche saranno prese in considerazione;
- la lista *_m* diventerà un sottoinsieme delle colonne della tabella, la lista *_m* sarà data dall'intersezione delle specie considerate e dai metaboliti presenti nell'ambiente.

Nelle iterazioni successive alla prima è necessario considerare anche i metaboliti prodotti come nutrienti che si aggiungono all'ambiente e che possono influire i metabo-

lismi delle specie presenti, dato che 15 dei metaboliti prodotti possono anche essere utilizzati.

Per la creazione della tabella in questione è stato scritto uno script in Python, presente nell'appendice C.5, in cui sono stati presi in input due dizionari aventi come chiavi gli id dei *type strain* considerati e in entrambi il valore di ogni chiave è rappresentato da una lista i cui elementi sono stringhe, per uno dei due la lista contiene i metaboliti utilizzati per l'altro dizionario invece la lista contiene i metaboliti prodotti. Il passo successivo è stato unire, senza doppioni, i valori contenuti nei due dizionari, ottenendo i 248 metaboliti di interesse, messi in ordine alfabetico così da rendere più facile la ricerca di un nutriente. Successivamente è stato creato un dizionario, da cui poi si potrà creare la tabella excel, avente come chiave gli id dei batteri e come valore una lista inizializzata con 248 zeri. Considerando in ordine un metabolita alla volta è stato visto se appartenesse agli utilizzati, andando così ad aggiornare la lista inserendo -1, o se appartenesse ai prodotti, andando così ad aggiornare la lista inserendo +1, nel caso non appartenga a nessuna delle due classi è stato mantenuto il valore 0.

Con la soluzione implementativa proposta si darebbe all'utente la possibilità di scegliere quali tra i 248 possibili nutrienti siano presenti nell'ambiente di crescita, potendo così formare dei terreni personalizzabili e variabili ogni simulazione. Con i dati ottenuti in questo elaborato non è possibile ricavare il growth rate dei microorganismi in questi ambienti. Su BacDive non sono riportate informazioni su nutrienti che possano risultare tossici, nella versione iniziale del simulatore si considera la possibilità per i batteri di poter incontrare nell'ambiente dei nutrienti che siano sfavorevoli rispetto alla loro evoluzione e sarebbe di fondamentale importanza trovare per le specie batteriche quali siano questi composti.

ID	Nome Specie
2853	Acetivibrio clariflavus
17248	Aliivibrio fischeri
17805	Alkaliflexus imshenetskii
13509	Allokutzneria albata
13222	Amycolatopsis azurea
132584	Anaerocolumna cellulosilytica
140747	Anaerotignum aminivorans
5416	Andreesenia angusta
133154	Butyricimonas faecihominis
18026	Caldicellulosiruptor acetigenus
18027	Caldicellulosiruptor owensensis
2568	Clostridium cellulovorans
140317	Clostridium tepidum
132404	Enterobacter bugandensis
2537	Gottschalkia acidurici
2601	Hathewaya histolytica
13290	Kibdelosporangium aridum subsp. aridum
58	Kozakia baliensis
158659	Lentzea soli
133321	Marimonas arenosa
158306	Monoglobus pectinilyticus
10742	Nocardia nova
506	Ornatilinea apprima
132998	Aquisalinus luteolus
158351	Proteiniborus indolifex
12800	Pseudomonas aeruginosa
2760	Ruminiclostridium cellulolyticum
13444	Saccharopolyspora erythraea
16158	Streptomyces eurocidicus
15261	Streptomyces griseochromogenes
15391	Streptomyces lydicus
294	Tolumonas auensis
166205	Periweissella cryptocerci

Tabella 5.4: Tabella con le 33 *type strain* considerate

Capitolo 6

Conclusioni

Con questo elaborato si propone di illustrare lo stato dell'arte delle banche di dati contenenti informazioni su terreni di coltura e sui metabolismi di svariate specie batteriche, pensando anche ad una possibile proposta implementativa da inserire in un simulatore per comunità microbiche, come potrebbe essere BactLife.

Inizialmente è stata analizzata la valenza biologica, ossia quanto siano di fondamentale importanza le comunità microbiche, in particolare nel corpo umano. Sono stati, quindi, riportati gli aspetti cardine, facendo presente che la nostra conoscenza della materia, al momento, è limitata ed è uno studio in continua evoluzione. Successivamente è stata esaminata anche la reperibilità delle informazioni presenti in letteratura per poter effettuare simulazioni con dati relativi ai metabolismi e alla composizione dell'ambiente che siano veritieri, dato che, come illustrato nel capitolo 3 e nel capitolo 4, non è elementare trovare dati che possano essere facilmente analizzati e implementati.

Per lo studio dei terreni di coltura si è preso in considerazione il database MediaDive, contenente le informazioni su composizione dei terreni e microorganismi che possano crescervi all'interno. Per i dati relativi ai metabolismi è stato studiato in primis BacDive, in cui sono presenti 19.313 *type strain*, ma non si possono trovare per ognuno di essi i metaboliti prodotti e/o utilizzati. É stato deciso di considerare solo i batteri aventi almeno 6 metaboliti prodotti e che abbiano informazioni sui metaboliti utilizzati; solo 33 *type strain* hanno tali caratteristiche (riportati nella tabella 5.4). Visto che entrambe le banche dati appena citate sono state progettate dalla *DMSZ*, collezione tedesca di microrganismi e colture cellulari, si è pensato che i metaboliti utilizzati dalle principali specie batteriche siano presenti tra gli ingredienti che formano i terreni di coltura descritti in MediaDive. In tal modo, la composizione dell'ambiente di crescita

dovrebbe risultare compatibile con i nutrienti necessari ai microorganismi per la loro evoluzione. I dati ottenuti dalle analisi effettuate mostrano che solo 107 su 197 della totalità dei metaboliti utilizzati dalle 33 specie sono presenti come ingredienti dei terreni. La povertà di dati presente nell'intersezione porta alla conclusione che non è possibile utilizzare solo i dati di MediaDive se si vogliono considerare i metabolismi riportati su BacDive.

Per poter avere una panoramica maggiore sulle informazioni presenti per i metabolismi dei batteri sono stati studiati due repository di una banca dati, chiamata Virtual Metabolic Human (VMH). AGORA è la prima versione pubblicata, in cui sono presenti 818 modelli, ognuno rappresentante un microorganismo. Ciascun modello è contenuto in un file .mat dove sono riportate moltissime informazioni, ad esempio ogni reazione compiuta. AGREDA è una versione aggiornata di AGORA in cui sono state inserite 179 vie metaboliche dei composti fenolici, le cui informazioni sono state prese da i-Diet, e si differenzia rispetto al primo repository per avere un unico file dentro al quale si possono trovare le caratteristiche degli 818 batteri. Per redigere un bilancio finale dei metaboliti che vengono utilizzati e prodotti, bisogna fare un'analisi tra tutte le reazioni che avvengono per ogni via metabolica attivata.

Nel paragrafo 5.2 di questo elaborato è stata esposta una possibile scelta implementativa: si è pensato di creare una tabella, una sua porzione è presentata in figura 5.4, con i 33 *type strain* ricavati da BacDive che abbiano più di 6 metaboliti prodotti e siano noti i loro metaboliti utilizzati. É stata fatta l'unione senza doppioni tra i metaboliti utilizzati e prodotti da tutte le specie, ottenendo un totale di 248 metaboliti. Nella tabella sono presenti 34 colonne totali, dove la prima indica tutti i metaboliti presenti e le altre 33 i metabolismi dei batteri, e 249 righe, in cui nella prima sono riportati gli id di tutte le specie e le restanti 248 sono tutti i metaboliti. Le celle contengono tre valori possibili:

- 0 se il nutriente non è di interesse per la specie;
- +1 se il metabolita viene prodotto;
- -1 se il metabolita viene utilizzato;

La scelta implementativa proposta ha come vantaggi di avere un valore scientifico e di essere facilmente implementabile permettendo un rapido aggiornamento dei simulatori. Inoltre, l'utente potrebbe scegliere di volta in volta i nutrienti che comporranno l'ambiente di crescita tra i 248 possibili metaboliti.

Risulta evidente, dunque, che con la soluzione proposta si possano effettuare simulazioni conformi al reale sviluppo di comunità microbiche. Tuttavia, è necessario

evidenziare la presenza di alcuni aspetti critici, tra cui:

- i metabolismi non hanno dei coefficienti stechiometrici, bensì solo 0/1/-1; questo problema potrebbe essere risolto andando a fare un'analisi, computazionalmente molto più onerosa rispetto a quanto fatto, grazie a COBRApy sul modello di AGREDA;
- su BacDive non sono presenti informazioni su nutrienti potenzialmente tossici per la specie, dunque sarebbe auspicabile effettuare delle ricerche affinché venga implementato questo dato nella tabella già esistente;
- ad ora non sono presenti terreni preimpostati che l'utente possa direttamente selezionare. Si potrebbero, quindi, cercare nella tabella quali tra i nutrienti formano ambienti di crescita noti. Ciò permetterebbe all'utente di optare per un terreno standard già impostato, senza dover scegliere tutte le singole componenti.

Una volta effettuate le implementazioni proposte, sarebbe utile effettuare ulteriori modifiche sul simulatore BactLife:

- introdurre un bolo alimentare che attraversa l'ambiente di sviluppo;
- espandere l'ambiente in un due, e successivamente tre, dimensioni;
- validare i risultati ottenuti con delle sperimentazioni in laboratorio.

Bibliografia

[1] A. Calzavara, Implementazione di un simulatore per comunità microbiche basato su un modello multi-agente. Tesi triennale in Ingegneria Biomedica. Università degli Studi di Padova, 2022.

[2] A. Lucchiari, Bactlife: simulatore per comunità batteriche - sviluppo del pacchetto Python.Tesi triennale in Ingegneria Biomedica. Università degli Studi di Padova, 2022.

[3] S. Rebecca, Bactlife: simulatore per comunità batteriche - sviluppo di interfaccia grafica in Dash. Tesi triennale in Ingegneria Biomedica. Università degli Studi di Padova, 2022.

[4] Associations of healthy food choices with gut microbiota profiles — Elsevier Enhanced Reader (2021). Available at: https://doi.org/10.1093/ajcn/nqab077

[5] Rowland, I. et al. (2018) 'Gut microbiota functions: metabolism of nutrients and other food components', European Journal of Nutrition, 57(1), pp. 1–24. Available at: https://doi.org/10.1007/s00394-017-1445-8.

[6] BacDive in 2022: the knowledge base for standardized bacterial and archaeal data Lorenz Christian Reimer, Joaquim Sardà Carbasse, Julia Koblitz, Christian Ebeling, Adam Podstawka, Jörg Overmann Nucleic Acids Research; database issue 2022.

[7] Koblitz, J., Halama, P., Spring, S., Thiel, V., Baschien, C., Hahnke, R., Pester, M., Overmann, J. and Reimer, L. (2022) MediaDive: the expert-curated cultivation media database. Nucleic Acids Research http://dx.doi.org/10.1093/nar/gkac803

[8] Magnusdottir, S., Heinken, A., Kutt, L., Ravcheev, D.A., Bauer, E., Noronha, A., Greenhalgh, K., Jager, C., Baginska, J., Wilmes, P., Fleming, R.M.T., Thiele,

I., ”Generation of genome-scale metabolic reconstructions for 773 members of the human gut microbiota”, Nature Biotechnology, 35(1):81-89 (2017). https://www.nature.com/articles/nbt.3703

[9] Blasco, T., Pérez-Burillo, S., Balzerani, F., Hinojosa-Nogueira, D., Lerma-Aguilera, A., Pastoriza, S., ... & Planes, F. J. (2021). An extended reconstruction of human gut microbiota metabolism of dietary compounds. Nature communications, 12(1), 1-12.

[10] Towards a deeper understanding of microbial communities: integrating experimental data with dynamic models — Elsevier Enhanced Reader, 2021. Available at: https://doi.org/10.1016/j.mib.2021.05.003.

[11] Bucci, V. et al. (2016) ‘MDSINE: Microbial Dynamical Systems INference Engine for microbiome time-series analyses’, Genome Biology, 17(1), p. 121. Available at: https://doi.org/10.1186/s13059-016-0980-6.

[12] Stein, R.R. et al. (2013) ‘Ecological Modeling from Time-Series Inference: Insight into Dynamics and Stability of Intestinal Microbiota’, PLoS Computational Biology, 9(12), p. e1003388. Available at: https://doi.org/10.1371/journal.pcbi.1003388.

[13] Ives, A.R. et al. (2003) ‘Estimating Community Stability and Ecological Interactions from Time-Series Data’, Ecological Monographs, 73(2), pp. 301–330. Available at: https://doi.org/10.1890/0012-9615(2003)073[0301:ECSAEI]2.0.CO;2.

[14] Gibbons, S.M. et al. (2017) ‘Two dynamic regimes in the human gut microbiome’, PLOS Computational Biology, 13(2), p. e1005364. Available at: https://doi.org/10.1371/journal.pcbi.1005364.

[15] Ridenhour, B.J. et al. (2017) ‘Modeling time-series data from microbial communities’, The ISME Journal, 11(11), pp. 2526–2537. Available at: https://doi.org/10.1038/ismej.2017.107.

[16] Jong, S. C. (Shung-Chang), 1936-2020. ATCC Names of Industrial Fungi. Rockville, Md. :American Type Culture Collection, 1994.

[17] *Pseudomonas aeruginosa* in BacDive https://bacdive.dsmz.de/pdf-view/12800?doi=doi%3A10.13145%2Fbacdive12800.20230509.8

[18] Noronha et al., ”The Virtual Metabolic Human database: integrating human and gut microbiome metabolism with nutrition and disease”, Nucleic Acids Research (2018); https://academic.oup.com/nar/advance-article/doi/10.1093/nar/gky992/5146204

[19] Vlassis N, Pacheco MP, Sauter T (2014) Fast Reconstruction of Compact Context-Specific Metabolic Network Models. PLoS Comput Biol 10(1): e1003424.

[20] Kenefake, D. et al. (2022) ‘An improved algorithm for flux variability analysis’, BMC Bioinformatics, 23(1), p. 550. Available at: https://doi.org/10.1186/s12859-022-05089-9.

[21] Navid A. A Beginner’s Guide to the COBRA Toolbox. Methods Mol Biol. 2022;2349:339-365. doi:10.1007/978-1-0716-1585-0_15.PMID: 34719002.

[22] Ebrahim, A. et al. (2013) ‘COBRApy: COnstraints-Based Reconstruction and Analysis for Python’, BMC Systems Biology, 7(1), p. 74. Available at: https://doi.org/10.1186/1752-0509-7-74.

Appendice A

MediaDive

Script per ottenere la numerosità dei terreni a cui un terreno appartiene, il dizionario in questione è poi ordinato in ordine decrescente.

```
import json
import urllib.request

url = 'https://mediadive.dsmz.de/rest/ingredient/'

file = open('ingredients_0.json') #apro il file json contenente alcune informazioni sugli ingredienti
data = json.load(file)

ingredienti = dict()

id = []
name = []

len_data = len(data)

for i in range(len_data):
    id.append(data[i]['id']) #creo una lista contenente tutti gli id degli ingredients
    name.append(data[i]['name']) #creo una lista coi nomi

for j in range(len_data):
    new_url = url + str(id[j]) #creo ogni volta un url con l'id dei vari ingredienti
    webUrl = urllib.request.urlopen(new_url)
    info = webUrl.read()
```

```
24      info = info.decode('utf-8')
25      info = info.replace('null','None') #nelle pagine online e'
        presente la scritta null per indicare un'informazione non
        presente, se venisse lasciata cosi' non permetterebbe di
        trasformarlo in un dizionario quindi e' stato sostituito null
        con None
26      my_dict = eval(info) #serve per rendere la stringa ottenuta
        un dizionario
27      media = my_dict['data']['media']
28      ingredienti[str(name[j])] = len(media) #dizionario contente
        come chiavi i nomi degli ingredienti e come valori le loro
        numerosita'
29
30  sort = dict(sorted(ingredienti.items(), key=lambda x: x[1],
        reverse=True)) #ordino il dizionario in ordine decrescente
31  tf = open("sortedIngredientes.json", "w")
32  json.dump(sort,tf)
33  tf.close()
```

Listing A.1: Creazione file .json con dizionario ingredientes e quantità di terreni in cui sono presenti.

Script per ottenere i sinonimi dei nomi degli ingredienti.

```
import json
import urllib.request

url = 'https://mediadive.dsmz.de/rest/ingredient/'

file = open('ingredients_0.json') #apro il file json contenente alcune informazioni sugli ingredienti
data = json.load(file)

syn = list()

id = []
name = []

len_data = len(data)

for i in range(len_data):
    id.append(data[i]['id']) #creo una lista contenente tutti gli id degli ingredients
    name.append(data[i]['name']) #creo una lista coi nomi

for j in range(len_data):
    new_url = url + str(id[j]) #creo ogni volta url con l'id dei vari ingredienti
    print(id[j])
    webUrl = urllib.request.urlopen(new_url)
    info = webUrl.read()
    info = info.decode('utf-8')
    info = info.replace('null','None') #nelle pagine online c'e' scritto null per indicare un'informazione non presente, se venisse lasciato cosi' non permetterebbe di trasformarlo in un dizionario quindi sostituisco null con None
    my_dict = eval(info) #serve per rendere la stringa ottenuta un dizionario
    list_syn = my_dict['data']['synonyms'] #lista dei sinonimi di un singolo ingrediente
    syn.append(name[j])
    for k in range(len(list_syn)):
        if list_syn[k] in syn:
            syn = syn
        else:
            syn.append(list_syn[k])

```

```
print(len(syn))

tf = open("REALsortedIngredientes.json", "w")
json.dump(syn,tf)
tf.close()
```

Listing A.2: Sinonimi ingredientes.

Script per contare i batteri che possono crescere in un terreno.

```
import json
import urllib.request

url = 'https://mediadive.dsmz.de/rest/medium-strains/'

terreni = dict()

file = open('media_0.json')

data = json.load(file)

len_data = len(data)

id = []
name = []

for i in range(len_data):
    ins = str(data[i]['medium_id'])
    id.append(ins) #creo una lista contenente tutti gli id degli ingredients
    name.append(data[i]['name']) #creo una lista coi nomi

for j in range(len_data):
    new_url = url + id[j] #creo ogni volta url con l'id dei vari ingredienti
    try:
        webUrl = urllib.request.urlopen(new_url)
        info = webUrl.read()
        info = info.decode('utf-8')
        info = info.replace('null','None') #nelle pagine online e' presente la scritta null per indicare un'informazione non presente, se venisse lasciata cosi' non permetterebbe di trasformarlo in un dizionario quindi e' stato sostituito null con None
        my_dict = eval(info) #serve per rendere la stringa ottenuta un dizionario
        terreni[str(name[j])] = len(my_dict['data'])
    except:
        terreni[str(name[j])] = 0

sort = dict(sorted(terreni.items(), key=lambda x: x[1], reverse=True)) #ordino il dizionario in ordine decrescente
tf = open("sortedTerreni.json", "w")
```

```
36 json.dump(sort,tf)
37 tf.close()
```

Listing A.3: Batteri in grado di crescere in un terreno.

Appendice B

BacDive

Esempio di BacDive su come si utilizzano le API.

```
## import package
import bacdive

## initialize BacDive client
client = bacdive.BacdiveClient('test@test.de', 'password')

## search with a BacDive ID
client.search(id="5621")

## retrieval of data for the strain previously searched
for strain in client.retrieve():
    print(strain)

## search with more than one BacDive ID
client.search(id="5621;138170")

## retrieval of data for the strains previously searched
## (print only general information)
for strain in client.retrieve():
    print(strain["General"])

## search with a genus name
client.search(taxonomy="Myroides")

## search with a species name
client.search(taxonomy="Myroides odoratus")

## retrieval of data for the strains previously searched
```

```
## with filter -> retrieve only the data for culture collection no.
for strain in client.retrieve(["culture collection no."]):
    print(strain)

## search with culture collection number
client.search(culturecolno="DSM 100861")

## retrieval of data for the strain previously searched
## with filter -> retrieve only the data for "family"
for strain in client.retrieve(["family"]):
    print(strain)

## retrieval of data for the strain previously searched
## with filter -> retrieve only the data for "full scientific name" and "culture medium"
for strain in client.retrieve(["full scientific name","culture medium"]):
    print(strain)

## Go through DSM numbers and find the respective BacDive IDs.
for i in range(1,21):
    DSM_num="DSM "+str(i)
    print(DSM_num, end="\t")
    result=client.search(culturecolno=DSM_num)
    if result:
        for strain in client.retrieve(["BacDive-ID"]):
            print(list(strain)[0])

## To which species do the BacDive strains of the genus Myroides belong?
client.search(taxonomy="Myroides")
species=[]
for strain in client.retrieve(["species"]):
    species.append(strain[list(strain)[0]][0]["species"])
from collections import Counter
print(Counter(species))
```

Listing B.1: Tutorial BacDive.

Appendice C

Aspetti implementativi

File utilizzato per ottenere liste e dizionari dei metaboliti utilizzati e prodotti dai batteri presenti su BacDive.

```
import bacdive
import json

client = bacdive.BacdiveClient( 'test@test . de      ,
    password     ')#inserire email e password

file = open('id_bacdive.json')
data = json.load(file)

len_data = len(data)

met_ut = list()
met_prod  = list()
uti = dict()
prod = dict()

for j in range(len_data):
    id = str(data[j])
    client.search(id=id) #metto come parametro ogni id dei
    batteri type strain presenti in bacdive
    for strain in client.retrieve('Physiology and metabolism'):
        for metabolite in client.retrieve('metabolite utilization
    '): #vado a ricercare i metaboliti utilizzati da quel batterio
            id_uti =list()
            try: #questo serve perch  non tutti i batteri hanno
    le informazioni riguardanti i metaboliti utilizzati
```

```
23                length = len(metabolite[id][0]['metabolite
    utilization'])
24            except:
25                length = 0
26                len_new = 0
27            new = list() #serve per contare la quantit  di nuovi
     metaboliti utilizzati
28            for i in range(length): #uso come parametro length
    perch  nel caso in cui ci sia pi  di un metabolita
    utilizzato la struttura prelevata da internet contiene una
    lista in cui ogni valore    un dizionario
29                if type(metabolite[id][0]['metabolite utilization
    ']) is dict: #serve nel caso in cui si abbia un solo
    metabolita utilizzato si ha un dizionario e non una lista
30                    met_ut.append(metabolite[id][0]['metabolite
    utilization']['metabolite']) #aggiorno la lista dei metaboliti
     utilizzati
31                    met_ut = set(met_ut) #tolgo eventuali
    doppioni
32                    met_ut = list(met_ut) #dopo aver utilizzato
    set non ho pi  una lista quindi riporto l'istanza met_ut ad
    una lista
33                    uti[id] = 1 #aggiorno il dizionario, ho messo
     uguale a uno perch  in questa parte di codice sto
    considerando i batteri che hanno un solo metabolita utilizzato
34                else:
35                    new.append(metabolite[id][0]['metabolite
    utilization'][i]['metabolite'])  #la lista new    una lista
    che ogni volta che si aggiorna il ciclo for viene risettata ad
     una lista vuota, ed ogni volta ci vengono aggiunti i
    metaboliti utilizzati da quel batterio
36                    met_ut.append(metabolite[id][0]['metabolite
    utilization'][i]['metabolite']) #aggiorno e controllo e che
    non ci siano doppioni nella lista dei metaboliti utilizzati
37                    met_ut = set(met_ut)
38                    new = set(new) #tolgo eventuali doppioni
    anche dalla lista new
39                    new = list(new)
40                    len_new = len(new) #   il parametro che mi
    indica la quantit  di metaboliti utilizzati dal
    microorganismo
41                    met_ut = list(met_ut)
42                    uti[id] = len_new #aggiorno il dizionario
```

```
43          for production in client.retrieve('metabolite production'
    ): #analizzo i metaboliti prodotti
44              for k in range(len(production[id])):
45                  try: #il try    necessario perch  non tutti i
    batteri hanno le informazioni riguardani i metaboliti prodotti
46                      length1 = len(production[id][k]['metabolite
    production']) #questo parametro serve perch  si avr  una
    lista contente in ogni posizione un dizionario con le
    informazioni sul metabolita, nel caso ce ne sia uno solo non
    si ha una lista ma subito un dizionario
47                  except:
48                      length1 = 0
49                  for q in range(length1):
50                      if type(production[id][k]['metabolite
    production']) is dict: #serve nel caso in cui si abbia un solo
     metabolita prodotto, si ha un dizionario e non una lista
51                          if production[id][k]['metabolite
    production']['metabolite'] in met_prod: #considero il caso in
    cui il metabolita sia gi  presente nella lista
52                              met_prod = met_prod
53                              prod[id] = 1 #aggiorno il dizionario
54                          else:
55                              met_prod.append(production[id][k]['
    metabolite production']['metabolite']) #aggiorno la lista
56                              prod[id] = 1 #aggiorno il dizionario
57                      else:
58                          if production[id][k]['metabolite
    production'][q]['metabolite'] in met_prod :
59                              met_prod = met_prod
60                          else:
61                              met_prod.append(production[id][k]['
    metabolite production'][q]['metabolite'])
62                              prod[id] = length1
63
64 #creo 4 file .json dove verrano inseriti i due dizionari e le due
     liste
65
66 tf = open("Met_utilizatoin.json", "w")
67 json.dump(uti,tf)
68 tf.close()
69
70 jo = open("Met_uti_list.json", "w")
71 json.dump(met_ut,jo)
72 jo.close()
```

```
73
74 so = open("Met_production.json", "w")
75 json.dump(prod,so)
76 so.close()
77
78 n = open("Met_prod_list.json", "w")
79 json.dump(met_prod,n)
80 n.close()
```

Listing C.1: Dizionari e liste dei metaboliti utilizzati e prodotti dei batteri di BacDive.

Memorizzazione batteri sogliati con dati sui metaboliti utilizzati. Boxplot in figura 5.3.

```
import json
import matplotlib.pyplot as plt
from numpy import var

file_prod = open('id_bact_sogliati_prod_4.json')
data = json.load(file_prod)

file_prod = open('id_bact_sogliati_prod_6.json')
data2 = json.load(file_prod)

file_uti = open('sorted_Met_utilization.json')
uti = json.load(file_uti)

somma = 0
non_p = 0
met = []
id1 = []

for i in range(len(data)):
    try: #cerca se nel dizionario dei met utilizzati (chiave gli id e valore intero rappresentante metaboli utilizzati) ci sono gli id dei batteri sogliati
        somma +=  uti[data[i]]
        met.append(uti[data[i]])
        id1.append(data[i]) #id batteri sogliati con anche dati su metaboliti utilizzati
    except: #se non presenti nel dizionario viene aggiornato non_p che conta quanti id non ci sono
        somma += 0
        non_p += 1
        met = met

media = somma / len(data) #calcola media dei batteri utilizzati
varianza = var(met) #calcola varianza dei batteri utilizzati

somma2 = 0
non_p2 = 0
met2 = []
id2 = []
for i in range(len(data2)):
    try: #cerca se nel dizionario dei met utilizzati (chiave gli id e valore intero rappresentante metaboli utilizzati) ci sono gli id dei batteri sogliati
```

```
38          somma2 +=  uti[data2[i]]
39          met2.append(uti[data2[i]])
40          id2.append(data2[i]) #id batteri sogliati con anche dati
    su metaboliti utilizzati
41      except: #se non presenti nel dizionario viene aggiornato
    non_p2 che conta quanti id non ci sono
42          somma2 += 0
43          non_p2 += 1
44          met2 = met2
45
46  media2 = somma2 / len(data2) #calcola media dei batteri
    utilizzati
47  varianza2 = var(met2) #calcola varianza dei batteri utilizzati
48
49  print(len(id1))
50  print(media)
51  print(non_p)
52  print(varianza)
53
54  print(len(id2))
55  print(media2)
56  print(non_p2)
57  print(varianza2)
58
59  met3 = [met,met2]
60  plt.boxplot(met3,labels=['4','6'])
61
62  plt.xlabel('Filtro sui metaboliti prodotti')
63  plt.ylabel('Metaboliti utilizzati')
64  plt.show()
65
66  tf = open("id_soglia4_uti.json", "w")#file contenente gli id dei
    batteri che hanno anche informazioni sui metaboliti utilizzati
67  json.dump(id1,tf)
68  tf.close()
69
70  tf2 = open("id_soglia6_uti.json", "w")#file contenente gli id dei
     batteri che hanno anche informazioni sui metaboliti
    utilizzati
71  json.dump(id2,tf2)
72  tf.close()
```

Listing C.2: Metaboliti utilizzati per i batteri sogliati e boxplot.

Codice con cui sono stati ottenuti i dati relativi all'intersezione tra i metaboliti prodotti e utilizzati.

```
import bacdive
import json

client = bacdive.BacdiveClient('niccolo.venturinidegliesposti@studenti.unipd.it', 'NvDE01uNi19')

file_4 = open('id_soglia4_uti.json')
data4 = json.load(file_4)

file_6 = open('id_soglia6_uti.json')
data6 = json.load(file_6)

#liste che conterranno i metaboliti prodotti e utilizzati per le due diverse soglie
prod4 = list()
uti4 = list()

prod6 = list()
uti6 = list()

for u in range(len(data4)):
    id =data4[u]

    met_prod4 = [] #lista che conterr  i metaboliti prodotti di un singolo batterio
    met_uti4 = [] #lista che conterr  i metaboliti utilizzati di un singolo batterio
    list_inters4 = []
    client.search(id=id)
    for strain in client.retrieve('Physiology and metabolism'):
        for production in client.retrieve('metabolite production'): #cerco le informazioni sui metaboliti prodotti
            for k in range(len(production[id])):
                try:
                    length1 = len(production[id][k]['metabolite production']) #numero di metaboliti prodotti da un batterio
                except:
                    length1 = 0
                for i in range(length1):
                    met_prod4.append(production[id][k]['metabolite production'][i]['metabolite']) #aggiungo alla lista
```

```
    i metaboliti prodotti da un batterio
35
36     for p in range(len(met_prod4)): #creazione lista dei
   metaboliti prodotti
37         if met_prod4[p] in prod4:
38             prod4 = prod4
39         else:
40             prod4.append(met_prod4[p])
41
42     for strain in client.retrieve('Physiology and metabolism'):
43         for utilization in client.retrieve('metabolite
   utilization'): #cerco le informazioni sui metaboliti
   utilizzati
44                 try:
45                      length2 = len(utilization[id][0]['metabolite
    utilization']) #numero di metaboliti utilizzati da un
   batterio
46                 except:
47                     length2 = 0
48                 for i in range(length2):
49                     met_uti4.append(utilization[id][0]['
   metabolite utilization'][i]['metabolite']) #aggiungo alla
   lista i metaboliti utilizzati da un batterio
50
51     for p in range(len(met_uti4)): #creazione lista dei
   metaboliti utilizzati
52         if met_uti4[p] in uti4:
53             uti4 = uti4
54         else:
55             uti4.append(met_uti4[p])
56
57 print(len(prod4))
58 print(len(uti4))
59
60 inter4 = list()
61
62 for i in range(len(uti4)): #ciclo for necessario per verificare
   le intersezioni tra metaboliti utilizzati e prodotti
63     if uti4[i] in prod4:
64         inter4.append(uti4[i])
65
66 print(len(inter4))
67
68 for u in range(len(data6)):
```

```
69      id =data6[u]
70
71      met_prod6 = [] #lista che conterr  i metaboliti prodotti di
    un singolo batterio
72      met_uti6 = [] #lista che conterr  i metaboliti utilizzati di
     un singolo batterio
73      list_inters6 = []
74      client.search(id=id)
75      for strain in client.retrieve('Physiology and metabolism'):
76          for production in client.retrieve('metabolite production'
    ): #cerco le informazioni sui metaboliti prodotti
77              for k in range(len(production[id])):
78                  try:
79                      length1 = len(production[id][k]['metabolite
    production']) #numero di metaboliti prodotti da un batterio
80                  except:
81                      length1 = 0
82                  for i in range(length1):
83                      met_prod6.append(production[id][k]['
    metabolite production'][i]['metabolite'])#aggiungo alla lista
    i metaboliti prodotti da un batterio
84
85      for p in range(len(met_prod6)): #creazione lista dei
    metaboliti prodotti
86          if met_prod6[p] in prod6:
87              prod6 = prod6
88          else:
89              prod6.append(met_prod6[p])
90
91      for strain in client.retrieve('Physiology and metabolism'):
92          for utilization in client.retrieve('metabolite
    utilization'):#cerco le informazioni sui metaboliti utilizzati
93                  try:
94                       length2 = len(utilization[id][0]['metabolite
     utilization']) #numero di metaboliti prodotti da un batterio
95                  except:
96                      length2 = 0
97                  for i in range(length2):
98                      met_uti6.append(utilization[id][0]['
    metabolite utilization'][i]['metabolite'])#aggiungo alla lista
     i metaboliti prodotti da un batterio
99
100     for p in range(len(met_uti6)): #creazione lista dei
    metaboliti utilizzati
```

```
101         if met_uti6[p] in uti6:
102             uti6 = uti6
103         else:
104             uti6.append(met_uti6[p])
105 
106 print(len(prod6))
107 print(len(uti6))
108 
109 inter6 = list()
110 for i in range(len(uti6)): #ciclo for necessario per verificare
    le intersezioni tra metaboliti utilizzati e prodotti
111     if uti6[i] in prod6:
112         inter6.append(uti6[i])
113 
114 print(len(inter6))
115 
116 tf4 = open("True_Global_intersezione_soglia4.json", "w")
117 json.dump(inter4,tf4)
118 tf4.close()
119 
120 tf6 = open("True_Global_intersezione_soglia6.json", "w")
121 json.dump(inter6,tf6)
122 tf6.close()
```

Listing C.3: Intersezione metaboliti prodotti e utilizzati

Metaboliti prodotti e utilizzati da tutti i *type strain* di BacDive presenti negli ingredienti di MediaDive.

```
import json

file = open('REALsortedIngredientes.json')
ing = json.load(file)

file = open('Met_prod_list.json')
prod = json.load(file)
no_prod = list()

file = open('Met_uti_list.json')
uti = json.load(file)
no_uti = list()

uti_prod = list()

#ciclo for per mettere tuttte le lettere di ogni ingrediente in
    maiuscolo
for q in range(len(ing)):
    ing[q] = str(ing[q])
    ing[q] = ing[q].upper()

#ciclo for per vedere quanti metaboli prodotti non sono tra gli
    ingredienti
for i in range(len(prod)):
    prod[i] = prod[i].upper()
    if prod[i] in ing:
        no_prod = no_prod
    else:
        no_prod.append(prod[i])

#ciclo for per vedere quanti metaboli utilizzati non sono tra gli
    ingredienti
for j in range(len(uti)):
    uti[j] = uti[j].upper()
    if uti[j] in ing:
        no_uti = no_uti
    else:
        no_uti.append(uti[j])

print(len(prod))
print(len(no_prod))
```

```
print(len(uti))
print(len(no_uti))
```

Listing C.4: Metaboliti di BacDive presenti in MediaDive

Script scritto per la creazione della tabella avente come colonne le 33 specie di BacDive con più di 6 metaboliti utilizzati e come righe i metaboliti prodotti e utilizzati.

```
import json
from tabulate import tabulate
import pandas as pd
import copy

with open('REALdict_uti_6.json') as file_uti:
    datau = json.load(file_uti)

with open('REALdict_prod_6.json') as file_prod:
    datap = json.load(file_prod)

with open('id_soglia6_uti.json') as file_bact:
    bact = json.load(file_bact)

list_uti = datau.values()
list_uti = list(list_uti)

list_prod = datap.values()
list_prod = list(list_prod)

#faccio l'unione senza intersezione degli utilizzati
uti = set()

for i in range(len(list_uti)):
    uti_set = set(list_uti[i]) | set(uti)
    uti = list(uti_set)

#faccio l'unione senza intersezione dei prodotti
prod = set()

for i in range(len(list_prod)):
    prod_set = set(list_prod[i]) | set(prod)
    prod = list(prod_set)

#creo la lista dei metaboliti che contiene sia gli utilizzati che
    i prodotti
met = set(uti) | set(prod)
met = list(met)

met.sort() #ordino in ordine alfabetico la lista

```

```
table = dict()
table['Metabolites'] = met

#creo un vettore contenente tanti zeri quanti sono il numero di
   metaboliti
lista = list()

for i in range(len(met)):
    lista.append(0)

#creo un dizionario che ha come chiavi gli id dei batteri e come
   valori un vettore che ha la stessa numerosit  dei metaboliti
for i in range(len(bact)):
    table[bact[i]] = lista

#sostituisco i valori nel dizionario, se un metabolita
   utilizzato inserisco -1, se    prodotto +1
for i in range(len(bact)):
    lista1 = copy.copy(lista) #shallow copoy of lista
    for j in range(len(met)):
        if met[j] in datap[str(bact[i])]:
            lista1[j] = 1
        elif met[j] in datau[str(bact[i])]:
            lista1[j] = -1
    table[bact[i]] = lista1

df = pd.DataFrame(table)

result = tabulate(table, headers="keys", tablefmt="fancy_grid")

#esporto la tabella in un file excel
df.to_excel("table.xlsx", index=False)

tf = open("REALmet_in_table.json", "w")
json.dump(met,tf)
tf.close()
```

Listing C.5: Creazione tabella excel coi dati utili di BacDive