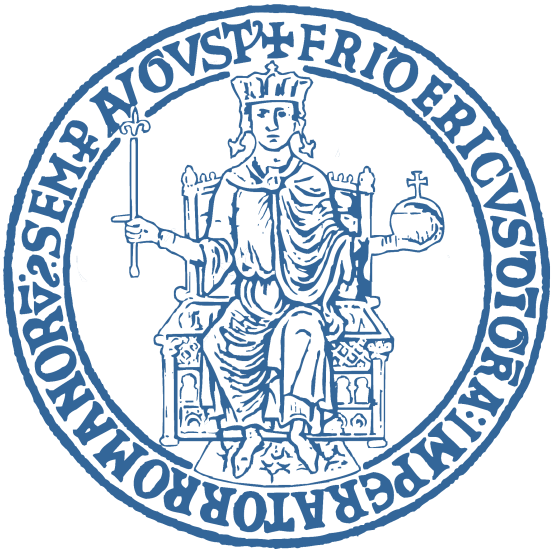
Università degli Studi di Napoli Federico II

Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione



Corso di Laurea Magistrale in Informatica

Una comparazione di algoritmi metaeuristici per il problema Repetition Free Longest Common Subsequence

Relatrice

Prof.ssa Paola Festa

Correlatore

Prof. Massimo Benerecetti

Candidato

Daniele Cuomo

Matricola

N97000270

Anno Accademico 2018/2019

Abstract

Repetition Free Longest Common Subsequence is a measure of similarity between pairs of strings, introduced by S. S. Adi et al. [1], proposed as an inference method for the search for homologous gene sequences. Calculating this measure is an APX-hard problem and, to date, an approximation algorithm with a constant performance ratio is not known.

This result justifies the use of combinatorial optimization techniques for problem solving, such as the *meta-heuristics*. The meta-heuristic frameworks are resolution methodologies that are abstracted from the specific problem, outlining research procedures within a generic solution space. This thesis work presents the realisation of three different meta-heuristics that exploit a new *greedy* selection criterion, whose goodness is verified experimentally. The algorithms are afterwards subjects of comparison with the current state of the art and with an approximation algorithm, for which a family of instances is exhibited which leads to the worst possible solutions.

The overall results show methods that can compute competitive solutions, in short waiting times. The latter is a fundamental requirement in application contexts where the amount of data is huge and/or the size of individual instances is hardly sustainable for a modern computer.

Sommario

Il *Repetition Free Longest Common Subsequence* è una misura di similarità tra coppie di stringhe, introdotta da S. S. Adi et al. [1], proposta come metodo di inferenza per la ricerca di sequenze geniche omologhe. Calcolare tale misura è un problema APX-hard e, ad oggi, non si è trovato un algoritmo di approssimazione con rapporto di prestazione costante.

Questo risultato giustifica l'impiego di tecniche di ottimizzazione combinatoria per la risoluzione del problema, come le *meta-euristiche*. I framework meta-euristici sono delle metodologie di risoluzione che si astraggono dal problema specifico, delineando dei procedimenti di ricerca all'interno di un generico spazio delle soluzioni. In questo elaborato di tesi si presenta l'istanziazione di tre diverse meta-euristiche che sfruttano un nuovo criterio di selezione *greedy*, la cui bontà è verificata sperimentalmente. Gli algoritmi sono successivamente oggetto di comparazione con l'attuale stato dell'arte e con un algoritmo di approssimazione, per il quale si esibisce una famiglia di istanze che porta alle soluzioni peggiori possibili.

I risultati complessivi mostrano dei metodi in grado di calcolare delle soluzioni competitive, in tempi di attesa esigui. Quest'ultimo è un requisito fondamentale in contesti applicativi dove la mole di dati è enorme e/o la taglia delle singole istanze è difficilmente sostenibile per un computer moderno.

Indice

Capitolo 1

Introduzione

1.1 Bioinformatica

Con il termine Bioinformatica si fa riferimento alla disciplina dedicata alla risoluzione di problemi biologici con il supporto di mezzi informatici. Gli obiettivi principali che si pone chi lavora in quest'ambito sono:

- Fornire modelli statistici validi per l'interpretazione dei campioni.
- Progettazione di banche dati che favoriscano l'integrazione di una conoscenza altrimenti frammentata.
- Generare strumenti matematici per l'analisi di sequenze e struttura del DNA, RNA e proteine al fine di comprenderne funzione ed evoluzione.

Quest'ultimo obiettivo porta spesso alla definizione di problemi di risoluzione non banale, per i quali risulta adeguato l'impiego di metodologie di Ottimizzazione Combinatoria. Esistono numerosi problemi (i più famosi si trovano sulla piattaforma Rosalind [2]) per i quali il calcolo di una soluzione ottimale risulta impraticabile. Spesso la mole di dati ottenuta dal sequenziamento[1] di un campione d'interesse risulta estremamente vasta e, anche per problemi trattabili, può essere

[1]Per sequenziamento si intende il processo di determinazione di una sequenza di nucleotidi

necessario adottare tecniche di approssimazione di una soluzione, in modo tale da contenere gli sforzi computazionali, sia in termini di tempo che occupazione di memoria.

Attualmente la comunità scientifica sta investendo le proprie risorse per problemi di *sequence alignment*, *protein folding* e *phylogenetic reconstruction*.

1.1.1 Sequence Alignment

Due sequenze si definiscono omologhe se sperimentalmente si dimostra che derivano da una sequenza ancestrale comune durante il corso dell'evoluzione. Ad esempio, due sequenze di proteine possono essere definite omologhe, se mostrano nei rispettivi organismi delle funzioni correlate. Quindi, l'omologia è un concetto dinamico, e le famiglie di sequenze omologhe conosciute al momento possono cambiare al crescere della sensibilità dei metodi.
Il primo passo verso l'inferenza dell'omologia è cercare la somiglianza tra sequenze. Se due sequenze date sono molto lunghe, non è facile decidere se sono simili o meno. Per vedere se sono simili, è necessario allinearle correttamente. Quando le sequenze si evolvono a partire da un antenato comune, gli elementi possono subire delle sostituzioni, ossia quando un'elemento viene rimpiazzato da un altro. Oltre alle sostituzioni, durante il corso dell'evoluzione le sequenze possono accumulare un numero di eventi di altri due tipi: inserzioni (quando nuovi elementi compaiono in una sequenza in aggiunta a quelli esistenti) e delezioni (quando alcuni elementi scompaiono). Pertanto, quando si cerca di produrre il miglior allineamento possibile tra due sequenze, occorre generalizzare tramite l'introduzione di spazi vuoti. La presenza di uno spazio vuoto in un allineamento rappresenta un evento di inserimento o delezione.
Ad esempio, siano TACCAGT e CCCGTAA due piccole sequenze di nucleotidi, un possibile allineamento sarebbe

T A C C A G T - -
C - C C - G T A A

Con l'introduzione dei gap gli allineamenti esistenti sono molteplici. Per capire quale sia l'allineamento ottimo occorre essere in grado di associare uno *score* alle combinazioni. Quando si stabilisce un metodo di assegnamento dello *score* occorre che questo sia biologicamente rilevante. In [3] viene introdotta una famosa implementazione del calcolo di un allineamento ottimo che fa uso della programmazione dinamica. Un approccio *divide et impera* è riportato come alternativa meno space-consuming in [4], requisito spesso necessario in questo campo.

1.1.2 Protein Folding

Uno dei problemi più importanti nella biologia molecolare è il problema del protein folding: data la sequenza aminoacidica di una proteina, qual è la struttura della proteina in tre dimensioni? Questo problema è importante poiché la struttura di una proteina fornisce una chiave per comprendere la sua funzione biologica.

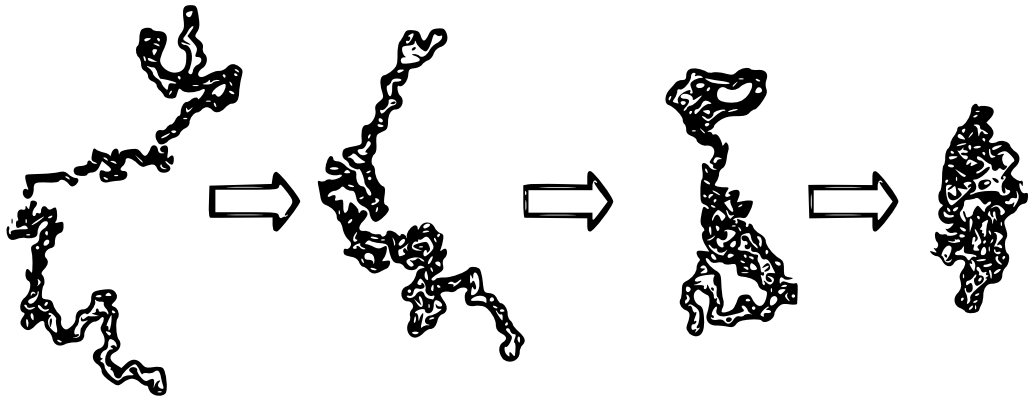


Figura 1.1: una proteina *unfolded* (sinistra) è detta inattiva. Affinchè una proteina sia attiva deve trovarsi nel suo stato nativo (destra).

È assunzione comune che la sequenza di amminoacidi contenga tutte le informazioni sulla struttura tridimensionale nativa di una molecola proteica in determinate condizioni fisiologiche.
Determinare la struttura tridimensionale di una proteina è molto difficile. I risultati più affidabili sono prodotti da approcci sperimentali costosi e che possono richiedere anni per produrre la struttura di una singola proteina.
Il numero di sequenze proteiche note è molto più grande del numero di strutture proteiche tridimensionali note e questo divario cresce costantemente come risultato di vari progetti di sequenziamento. Pertanto, i metodi matematici, statistici e computazionali che possono dare qualche indicazione metodologica stanno diventando sempre più importanti.

1.1.3 Phylogenetic Reconstruction

La filogenesi è la storia evolutiva di un gruppo di entità. Lo scopo principale della ricostruzione della filogenesi è di descrivere le relazioni evolutive in termini di antenati comuni. Queste relazioni sono rappresentate come un diagramma di ramificazione con rami uniti da nodi e che terminano con delle foglie (figura 1.2).

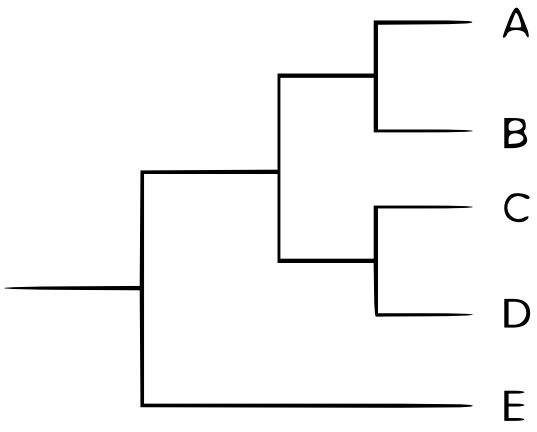


Figura 1.2: un albero filogenetico che mostra la relazione tra 5 specie generiche A, B, C, D ed E.

La metodologia filogenetica si basa sul presupposto che le sequenze usate per generare gli alberi sono omologhi. In seguito ad un'attenta fase di allineamen-

to delle sequenze, si possono effettuare le analisi filogenetiche. Ci sono diverse metodologie che possono essere implementate. Il processo decisionale del tipo di analisi dovrà tener conto del trade-off tra budget computazionale e precisione richiesta. Tra le principali tecniche si trovano le *distanze*. Dei metodi utili per la loro semplicità di calcolo, adeguati alla costruzione di una versione preliminare degli alberi [5]. Proprio in questa categoria ricade il problema oggetto di questa tesi e discusso nel prossimo capitolo.

1.2 Distanze Evolutive

Nel contesto della filogenesi si parla di distanza evolutiva come approccio alla ricostruzione filogenetica. Stimare la distanza tra due o più organismi richiede due step: specificare un modello evolutivo (il modo in cui gli organismi possono mutare), e decidere quale metrica usare. La metrica dovrà essere coerente con il modello adottato.

Due genomi, ad esempio, possono risultare apparentemente privi di alcun legame a causa di un fenomeno naturale detto *riarrangiamento del genoma* [6], ovvero una mutazione che colpisce il genoma nel suo processo evolutivo. Di conseguenza occorre considerare il caso in cui le sequenze presentano un gene comune ma localizzato in posizioni differenti.

Volendo formalizzare il problema in chiave informatica, si definisce un'istanza come una coppia di stringe (x, y) definite in un alfabeto Σ, si potrà dunque codificare una sequenza di nucleotidi tramite un simbolo $\sigma \in \Sigma$. Data una sequenza del tipo $(\sigma_1, \sigma_2, \ldots, \sigma_n)$, essa è detta sotto-sequenza comune in x e y se è una sequenza ordinata di elementi consecutivi, non necessariamente contigui, sia di x che di y. Formalmente è possibile individuare due sequenze di pedici $i_1 < i_2 < \cdots < i_n$ e $j_1 < j_2 < \cdots < j_n$ tali che $x_{i_1} = y_{j_1} = \sigma_1, x_{i_2} = y_{j_2} = \sigma_2, \ldots, x_{i_n} = y_{j_n} = \sigma_n$.

La prima metrica, riportata nella prossima sotto-sezione, definisce un criterio che

tenga conto delle caratteristiche biologiche descritte precedentemente.

1.2.1 Longest Common Subsequence

Sia $\texttt{LCS}(x, y)$ il problema di trovare una sotto-sequenza comune ad x e y di lunghezza massima. La formulazione matematica corrispondente consisterà in un problema di Programmazione Lineare Intera chiedendo di massimizzare la funzione obiettivo:

$$\begin{aligned} \max \quad & \sum_{z_{i,j} \in Z_\sigma} z_{i,j} \\ \text{s.a} \quad & z_{i,j} + z_{k,l} \leq 1, \ \forall z_{i,j}, z_{k,l} \text{ in stato di cross} \\ & z_{i,j} \in \{0, 1\}, \ \forall z_{i,j} \in Z \end{aligned}$$

Z consiste in un insieme di variabili di decisione. Il valore di una variabile di decisione $z_{i,j}$ indica se la coppia di lettere x_i e y_j è in soluzione o meno. Le coppie di interesse sono tutte e sole le (x_i, y_j) tale che $x_i = y_j$. Si definisce quindi una partizione di Z in sotto-insiemi $Z_\sigma = \{z_{i,j} \in Z \mid x_i = y_j = \sigma\}$. L'obiettivo è dunque quello di massimizzare il numero di variabili di decisione in soluzione, rispettando i vincoli di **cross**. Un cross è un predicato tra due variabili $z_{i,j}$ e $z_{k,l}$. Qualora risultasse che $i < k$ e $j > l$, o viceversa $i > k$ e $j < l$, le coppie corrispondenti provocano un cross e di conseguenza non possono stare nella stessa soluzione.

Il problema è risolvibile all'ottimo in tempo polinomiale sulla taglia dell'istanza. La soluzione più famosa opera secondo uno schema di programmazione dinamica, con una complessità computazionale pari a $\mathcal{O}(|x| \cdot |y|)$. Altri sforzi sono stati impiegati per abbassare i costi di memoria, spesso insufficiente per applicazioni reali, e/o per abbassare i tempi di calcolo, alcune proposte sono reperibili in [7].

1.2.2 Exemplar Distances

Le Exemplar Distances sono dei modelli nati dalla necessità di tenere traccia di altre peculiarità che caratterizzano il genoma. Durante il processo evolutivo possono avvenire delle duplicazioni geniche. Si genera quindi una *famiglia*, ovvero un insieme di geni formatisi a partire da un antenato comune, detto *esemplare*, tramite duplicazione [8]. La misura `LCS` diventa adeguata alle sola restrizione di istanze prive di duplicazioni. Un modo di approcciarsi al modello più generico può essere quindi tramite la definizione di un metodo per ricondurre la ricerca a questo spazio ristretto di istanze, individuando gli elementi esemplari ed eliminando le altre occorrenze. La scelta degli elementi dovrebbe seguire il principio di massima parsimonia, secondo il quale una stringa di soli esemplari sarà quella che minimizza il numero di mutazioni.
La definizione di nuovi modelli ha portato alla necessità di far evolvere la metrica `LCS`, aprendo nuove strade.

1.2.3 Metriche LCS

Si è visto che l'uso di `LCS`, come indice di correlazione, è adeguato solo in determinate circostanze, piuttosto restrittive. Ampliando il modello, occorre generalizzare la metrica, per preservare l'attendibilità biologica dello stimatore. Si è arrivati quindi alla definizione di un'intera famiglia di varianti a `LCS` . Famiglia di cui fa parte la metrica studiata durante l'attività di tesi. Si conclude quindi il capitolo con una stesura dei principali problemi affini ad `LCS`, ad eccezione del problema protagonista in questa tesi che è studiato in maniera approfondita nel capitolo 2. É importante osservare che `LCS` è generalizzabile ad istanze formate da un numero arbitrario di stringhe. Questa estensione risulta essere `NP`-hard e per questo motivo, per la definizione di metriche più precise, ci si limita a considerare istanze formate da due sole sequenze.

Exemplar LCS

L'input di ELCS consiste in una coppia (x, y) di sequenze definite sull'alfabeto $\Sigma_o \cup \Sigma_m$, $\Sigma_o \cap \Sigma_m = \emptyset$, dove Σ_o è l'insieme di simboli opzionali e Σ_m è l'insieme di simboli obbligatori (*mandatory*). L'output della metrica è una LCS s^* di x e y che contiene tutti i simboli obbligatori.
Il problema risulta APX-hard, come riportato in [9].
Esistono 4 diverse versioni di seguito elencate:

- $\texttt{ELCS}_1$: s^* ha esattamente un'occorrenza $\forall \sigma \in \Sigma_m$
- $\texttt{ELCS}_{\geq 1}$: s^* ha almeno un'occorrenza $\forall \sigma \in \Sigma_m$
- $\texttt{ELCS}_{1,\leq 1}$: s^* ha esattamente un'occorrenza $\forall \sigma \in \Sigma_m$ ed al più 1 occorrenza $\forall \sigma \in \Sigma_o$
- $\texttt{ELCS}_{\geq 1,\leq 1}$: s^* ha almeno un'occorrenza $\forall \sigma \in \Sigma_m$ ed al più 1 occorrenza $\forall \sigma \in \Sigma_o$

Constrained LCS

Il Constrained LCS [10] è così definito: date due stringhe x, y e un insieme di stringhe C, trovare una LCS s^* di x e y tale che ogni stringa in C sia una sottosequenza di s^*. Per un numero arbitrario di vincoli il problema è NP-hard, mentre CLCS con una singola stringa di vincoli è risolvibile in tempo polinomiale.

Doubly-Constrained LCS

Un'istanza è composta da sequenze definite in un alfabeto Σ e da una funzione $f_{max} : \Sigma \to \mathbb{N}$. Data una coppia di sequenze (x, y) e un'insieme C di stringhe. Si trovi un v^* LCS di x ed y che sia sotto-sequenza di tutte le stringhe in C e tale che, $\forall \sigma \in \Sigma$, $f(v^*, \sigma) \leq f_{max}(\sigma)$, con $f(v^*, \sigma)$ numero di occorrenze di σ in v^*.
In [11] si è dimostrata l'*hardness* per la classe di complessità APX.

Capitolo 2

Repetition Free Longest Common Subsequence

2.1 Definizione del Problema

Il Repetition Free Longest Common Subsequence è un problema introdotto in [1] da S. S. Adi et al. Così come l'Exemplar Longest Common Subsequence propone una misura che tenga conto dell'esistenza del fenomeno delle duplicazioni (1.2.2), anche in questo caso è proposta una variante che, anziché cercare i geni esemplari, si limita a valutare la somiglianza tra due genomi attraverso la loro sotto-sequenza più lunga, formata da al più un'occorrenza per ogni lettera dell'alfabeto, da cui la dicitura *repetition free*. Si osservi che il problema può essere interpretato come calcolo dell'*edit distance*[1] con l'eliminazione di occorrenze come unica operazione ammessa.

Formalmente, date due sequenze x ed y definite in un alfabeto Σ, il problema $\texttt{RFLCS}(x, y)$ consisterà nel trovare una sequenza w^* tale che

$$w^* = \arg\max\{|w| \mid w \text{ sotto-sequenza repetition free di } x \text{ e } y\}$$

[1] l'edit distance è una misura di somiglianza tra due stringhe che conta il numero di operazioni necessarie a trasformare una stringa in un'altra

La rispettiva formulazione matematica corrisponderà ad un problema di Programmazione Lineare Intera in cui è definito un insieme di variabili di decisione $Z = \{z_{i,j} \mid x_i = y_j\}$.

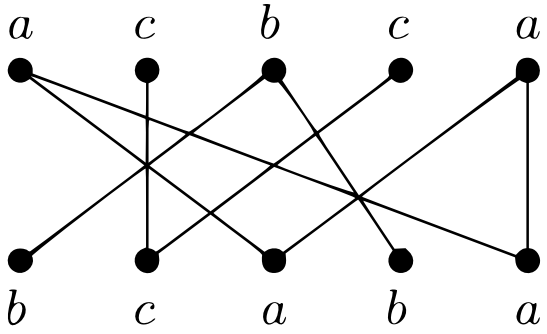


Figura 2.1: una possibile istanza con rappresentazione a grafo. Z corrisponde all'insieme di archi presenti.

Si definisce inoltre una partizione di Z in sotto-insiemi $Z_\sigma = \{z_{i,j} \mid x_i = y_j = \sigma\}$. Infine sia **cross** un predicato tra due variabili $z_{i,j}$ e $z_{k,l}$, risultante vero se si verifica che $i < k$ e $j > l$, o viceversa $i > k$ e $j < l$. Due variabili, per cui è verificata la condizione di cross, non possono stare nella stessa soluzione. Segue la formulazione:

$$
\begin{aligned}
\max \quad & \sum_{z_{i,j} \in Z} z_{i,j} \\
\text{s.a} \quad & \sum_{z_{i,j} \in Z_\sigma} z_{i,j} \leq 1, \ \forall \sigma \in \Sigma \\
& z_{i,j} + z_{k,l} \leq 1, \ \forall z_{i,j}, z_{k,l} \text{ in stato di cross} \\
& z_{i,j} \in \{0,1\}, \ \forall z_{i,j} \in Z
\end{aligned}
$$

La funzione obiettivo corrisponde alla ricerca di un numero di occerrenze in soluzione di cardinalità massima. Il primo insieme di vincoli impone che al più una variabile di decisione, per ogni elemento della partizione di Z, abbia valore 1. Con questo vincolo si garantisce che la stringa sia priva di ripetizioni di lettere. Il secondo ed ultimo insieme di vincoli impone che, per ogni coppia di variabili in stato di cross, al più una delle due venga aggiunta in soluzione.

É immediato osservare che il problema `LCS` (1.2.1) è molto simile. Di fatto è ottenibile tramite rilassamento per eliminazione del primo insieme di vincoli del problema `RFLCS` . Esistono diverse proposte in letteratura, riportate in 2.3, che fanno di questo rilassamento una componente fondamentale per la risoluzione del problema.

2.2 Risultati Teorici

2.2.1 Complessità Computazionale

Proposizione. *Il problema `RFLCS` è `APX`-hard.*

Il primo risultato qui riportato afferma che per il problema in esame sono verificate le condizioni di *hardness* per la classe di complessità `APX` [12]. La dimostrazione opera applicando una L-reduction [13] ad una restrizione di `RFLCS` a partire da un problema notoriamente `APX`-completo, il `MAX 2,3-SAT`, che è una particolare versione di `MAX 2-SAT`.

Sia V un insieme di variabili booleane. Indichiamo con $\bar{v}$ la negazione di una variabile v. Un letterale è un elemento di $V \cup \{\bar{v} \mid v \in V\}$. Una clausola è un insieme di letterali, ed è una k-clausola se ha k letterali. Un assegnamento per V è una funzione $h : V \to \{\texttt{T}, \texttt{F}\}$.

Un letterale l è `T` secondo h se, per qualche v in V, o $l = v$ e $h(v) = \texttt{T}$, o $l = v$ e $h(v) = \texttt{F}$. Una clausola è soddisfatta da un'assegnazione h se almeno uno dei suoi valori letterali è `T` secondo h. Il problema `MAX 2,3-SAT`(V, C) è così definito: dato un insieme C composto da delle 2-clausola su V, dove ogni letterale appare in al più 3 clausole in C, trovare un assegnamento per V che massimizza il numero di clausole soddisfatte in C. Questa variante di `MAX 2-SAT` è `APX`-completa [14]. Si suppone che nessuna clausola sia della forma $\{v, \bar{v}\}$, $\forall v \in V$. Dato un assegnamento h, si denota con $n(\texttt{MAX 2,3-SAT}(V, C), h)$ il numero di clausole in C che sono soddisfatte da h. Il valore ottimo di funzione

obiettivo sarà $\texttt{OPT}(\texttt{MAX 2,3-SAT}(V,C)) = \max_{\forall h}\{n(\texttt{MAX 2,3-SAT}(V,C),h)\}$. Una L-riduzione da `MAX 2,3-SAT` a `RFLCS` consiste in una coppia di funzioni calcolabili in tempo polinomiale (f,g) tali che, per due costanti positive fisse α e β, sono verificate le seguenti due condizioni:

C1 per ogni istanza (V,C) di `MAX 2,3-SAT`, $f(V,C) = (x,y)$ è un'istanza di `RFLCS` e $\texttt{OPT}(\texttt{RFLCS}(x,y)) \leq \alpha \cdot \texttt{OPT}(\texttt{MAX 2,3-SAT}(V,C))$;

C2 per ogni istanza (V,C) di `MAX 2,3-SAT` ed ogni sottosequenza repetition free w di x ed y, dove $(x,y) = f(V,C)$, si ha che $h = g(V,C,w)$ è un'assegnamento per V e $\texttt{OPT}(\texttt{MAX 2,3-SAT}(V,C)) - n(\texttt{MAX 2,3-SAT}(V,C),h) \leq \beta \cdot (\texttt{OPT}(\texttt{RFLCS}(x,y)) - |w|)$.

La proposizione segue immediatamente dal successivo teorema. Ai fini della dimostrazione si indica con $f(\sigma,v)$ frequenza di σ in una stringa v.

Teorema. *[1] La restrizione del problema **RFLCS** alle sole istanze (x,y) tali che $\forall\sigma \in \Sigma$ si verificano $f(\sigma,x) \leq 2$ e $f(\sigma,y) \leq 2$ è **APX**-completa.*

Dimostrazione. Nella sezione 2.3.1 si presentano delle m-approssimazioni per la risoluzione polinomiale del problema. La variabile m, limitando il problema alla restrizione in considerazione, assume sempre valore limitato superiormente da 2. Di conseguenza l'algoritmo ha rapporto di prestazione costante. Per dimostrarne l'`APX`-completezza resta quindi da dimostrare la riduzione a partire dal problema `MAX 2,3-SAT`, che avviene tramite L-riduzione.

Per un'istanza (V,C) di `MAX 2,3-SAT`, dove $V = \{v_1, v_2, \ldots, v_n\}$ e C è un insieme di 2-clausole su V, si descrive un'istanza $(x,y) = f(V,C)$ di `RFLCS`. Sia $\{c_1, c_2, \ldots, c_m\}$ un insieme di etichette distinte, una per ogni clausola di C. Per semplicità, si scrive c_i per riferirsi sia all'etichetta c_i che alla clausola la cui etichetta è c_i. Di conseguenza $C = \{c_1, c_2, \ldots, c_m\}$.

Per ogni letterale l, si denota con $s(l)$ una sequenza composta dalle (etichette delle) clausole in cui l è presente, prese secondo un ordine arbitrario. Quindi,

$\forall v \in V$ e dato un assegnamento h per V, la sequenza $s(v)$ contiene le clausole di C che sarebbero soddisfatte qualora $h(v) = \texttt{T}$ e le sequenze $s(\bar{v})$ contiene le clausole di C che sarebbero soddisfate se $h(v) = \texttt{F}$. Si osservi che, siccome si escludono le clausole della forma $\{v, \bar{v}\}$, allora $s(v)$ ed $s(\bar{v})$ non hanno simboli in comune. Inoltre, siccome ogni letterale l appare in al più 3 clausole, si ha $|s(l)| \leq 3$. Si definisce un ulteriore insieme di simboli $D = \{d_1, d_2, \ldots, d_k\}$, tale che $k = 6(n-1)$ e $D \cap C = \emptyset$. Le sequenze x ed y si costruiscono come segue:

$$x = s(v_1)s(\bar{v}_1)d_1 \cdots d_6 s(v_2)s(\bar{v}_2)d_7 \cdots d_{12} s(v_3)s(\bar{v}_3) \cdots d_k s(v_n)s(\bar{v}_n)$$

$$y = s(\bar{v}_1)s(v_1)d_1 \cdots d_6 s(\bar{v}_2)s(v_2)d_7 \cdots d_{12} s(\bar{v}_3)s(v_3) \cdots d_k s(\bar{v}_n)s(v_n)$$

L'alfabeto adottato è l'insieme $C \cup D$. Per definizione, gli insieme C e D sono disgiunti ed ogni simbolo di D occorre una volta sia in x che in y. Inoltre, siccome ogni clausola c in C ha due letterali, e, per ogni letterale l, la corrispondente sequenza $s(l)$ appare una volta sia in x che in y, segue che ogni simbolo c occorre due volte in x e altrettante in y.

Si noti che la costruzione può essere fatta in tempo polinomiale. Siccome tutte le clausole hanno 2 letterali, $n \leq 2m$, dove $n = |V|$ e $m = |C|$. Inoltre, ogni simbolo dell'alfabeto $C \cup D$ può apparire al più una volta in una sottosequenza repetition free di x ed y, quindi $\texttt{OPT}(\texttt{RFLCS}(x, y)) \leq m + 6(n-1) \leq 12m$. D'altro canto, si può facilmente fissare un assegnamento h per V tale che $n(\texttt{MAX 2,3-SAT}(V, C), h) \geq \frac{m}{2}$. Infatti, iterativamente, per $i = 1, 2, \ldots, n$ si definisce $C_i \subseteq C$ come $C_i = \{c \in C \mid v_i \in c \vee \bar{v}_i \in c\}$ e $C \leftarrow C \backslash C_i$; dopodichè si ponga $h(v_i) = T$, se v_i è più comune di $\bar{v}_i$ nelle clausole di C_i, $h(v_i) = \texttt{F}$ altrimenti. Si osservi che la C finale è vuota e h soddisfa almento $\frac{|C_i|}{2}$ clausole di C_i, per ogni i. Inoltre, siccome $\bigcup C_i$ è uguale all'insieme iniziale C, l'assegnamento h soddisfa almeno $\frac{m}{2}$ clausole di C. Quindi $\texttt{OPT}(\texttt{MAX 2,3-SAT}(V, C)) \geq \frac{m}{2}$. Unendo le due

disequazioni ottenute si conclude che

$$\texttt{OPT}(\texttt{RFLCS}(x, y)) \leq 24 \cdot \texttt{OPT}(\texttt{MAX 2,3-SAT}(V, C))$$

e **C1** è verificato con $\alpha = 24$.

Resta da dimostrare **C2**. Per farlo si mostra che esiste una sottosequenza repetition free w di x e y di lunghezza almeno $p = q + |D|$ se e solo se esiste un assegnamento h per V che soddisfa almeno $q = p - |D|$ clausole di C. Sia quindi w una sottosequenza repetition free di x e y di lunghezza p. Come primo passo si mostra una sottostringa repetition free z di lunghezza almeno p che contiene tutti i simboli in D. Per costruire z, a partire da w, si sostituiscono tutti i simboli allineati, che provocano un cross con regioni costituite da simboli consecutivi d_i, con un allineamento completo dei simboli d_i di quella regione. Il numero di d_i è scelto in maniera tale da assicurare che la sequenza ottenuta sia lunga almeno quanto l'originale. Successivamente si descrive un assegnamento h che soddisfa almeno $|z| - |D| \geq p - |D|$ clausole di C.

La procedura per costruire z a partire da w, così che z sia una sequenza che contiene tutti i simboli $d_1, d_2, \ldots, d_k$ (in questo ordine) e che sia almeno lunga quanto w, è la seguente. Sequenzialmente, per $i = 1, 2, \ldots, n-1$ si rimuove ogni simbolo di w proveniente da $s(v_i)s(\bar{v}_i)$ in x ($s(\bar{v}_i)s(v_i)$ in y) e si rimuova un simbolo di $s(\bar{v}_{i+1})s(v_{i+1})d_{6(i+1)-5} \cdots d_k s(\bar{v}_n)s(v_n)$ in y ($s(v_{i+1})s(\bar{v}_{i+1})d_{6(i+1)-5} \cdots d_k s(v_n)s(\bar{v}_n)$ in x rispettivamente), e si aggiunge $d_{6i-5}d_{6i-4}d_{6i-3}d_{6i-2}d_{6i-1}d_{6i}$, ovvero i simboli di D presenti tra $s(v_i)$ ed $s(v_{i+1})$ in x o y. Detta z la sequenza risultante, si aggiungano a z i simboli di D che non sono stati presi in considerazione nelle operazioni precedenti. Si osservi che $|s(v_i)s(\bar{v}_i)| \leq 6$, quindi ad ogni passo di sostituizione al più 6 simboli vengono rimpiazzati da altrettanti simboli di D (che non occorrono in w). Quindi z è una soluzione ammissibile tale che $|z| \geq |w|$.

Resta da stabile l'assegnamento h. Siccome z contiene tutti i simboli di D,

le altre porzioni di z sono sottosequenze di $s(v_i)s(\bar{v}_i)$ in x e $s(\bar{v}_i)s(v_i)$ in y, per ogni $i = 1, 2, \ldots, n$. Inoltre, siccome tutti i simboli in $s(v_i)$ differiscono da quelli in $s(\bar{v}_i)$, la sequenza z non presenta simultaneamente simboli di $s(v_i)$ e $s(\bar{v}_i)$. Quindi si definisce h come segue: $h(v_i) = \texttt{T}$ se z allinea un simbolo proveniente da $s(v_i)$, $h(v_i) = \texttt{F}$ altrimenti. Detto $g(V, C, w) = h$, si osservi che h soddisfa almeno $q = |z| - |D| \geq p - |D|$ clausole.

Per dimostrare l'altra direzione della doppia implicazione si consideri un assegnamento h per V che soddisfa q clausole di C. Sia w una soluzione ammissibile per (x, y) così ottenuta: per $i = 1, 2, \ldots, n$ si aggiungono a w i simboli che corrispondono alle clausole in $s(v_i)$ se $h(v_i) = \texttt{T}$, altrimenti si aggiungono i simboli che corrispondono alle clausole $s(\bar{v}_i)$. Fatto ciò, si eliminano le ripetizioni e si aggiungono a w, nella posizione corretta, tutti i simboli in D. w è una soluzione ammissibile tale che $|w| = q + |D|$. A questo punto è possibile affermare che il vincolo **C2** è verificato con $\beta = 1$. □

Questo risultato è sufficiente a garantire che, a meno che `P` = `NP`, non ha senso cercare un algoritmo esatto che risolva il problema in tempo polinomiale sulla taglia dell'istanza. Occorre quindi affrontare `RFLCS` tramite tecniche di ottimizzazione combinatoria, categoria in cui subentrano le meta-euristiche, ossia le tecniche adottate in questa tesi e discusse nel capitolo 3.

2.2.2 Riduzione al Maximum Indipendent Set

In [15] è proposto un metodo di riduzione che porta a valori di funzione obiettivo equivalenti per un'istanza del problema `RFLCS` e per la sua riduzione ad un'istanza del problema `MIS` (*Maximum Indipended Set*).

Nel problema del Maximum Indipendent Set, un'istanza consiste in un grafo $G = (V, E)$. L'obiettivo è quello di trovare un sotto-insieme indipendente $S \subseteq V$ di cardinalità massima. `MIS` è un problema largamente studiato e per il quale esistono euristiche che funzionano bene in contesti reali. Nello stesso articolo

si propone un'implementazione della riduzione seguita da un algoritmo greedy base per la risoluzione del problema ottenuto, scelta adottata come mero proof-of-concept. Segue il teorema:

Teorema. *[15] Le istanze di* `RFLCS` *sono polinomialmente riducibili ad istanze di* `MIS` *con valore di funzione obiettivo equivalente.*

Dimostrazione. La dimostrazione procede tramite costruzione di un grafo con un insieme di vertici $V = \{(i, j) \mid x_i = y_j\}$ a partire da un'istanza (x, y) di `RFLCS`. La soluzione per `MIS` consiste in un insieme $S \subseteq V$ ed ogni vertice $(i, j) \in S$ corrisponde ad un simbolo σ contenuto nella soluzione di `RFLCS`. Tramite costruzione di un insieme di archi E adeguato è possibile modellare i vincoli del problema su stringhe:

1. Vincoli di cross: per ogni coppia di nodi (i, j) e (k, l) in stato di cross, si collegano i nodi tramite un arco.

2. Vincolo di repetition free: si costruisce una clique per ogni sottoinsieme $V_\sigma = \{(i, j) \in V \mid x_i = y_j = \sigma\}$.

Siccome si cerca un massimo insieme indipendente, è immediato l'ottenimento di un sottosequenza comunque repetition free di lunghezza massima. □

Studiare le relazioni con il problema `MIS`, oltre ad essere utile per motivi pratici legati al fatto che il `MIS` conta molti anni di ricerca dedicata, è interessante in quanto è possibile dimostrare che, a meno che `P` = `NP` , si verifica `MIS` $\notin$ `APX` [16]. In altre parole non è possibile trovare un algoritmo di approssimazione con rapporto di prestazione costante per `MIS`.

2.2.3 Proprietà Aleatorie

Ulteriori sforzi sono stati fatti per analizzare i fenomeni aleatori del problema. In [17] si definisce la variabile aleatoria R_n per una tipologia d'istanza specifica $\mathcal{I} =$

(x, y) al problema `RFLCS`: R_n è il valore ottimo per le istanze di taglia $|x| = |y| = n$ generate tramite campionamento casuale uniforme. Si vuole quindi determinare il valore di $\mathbb{E}[R_n]$ in funzione di n e k, con k grandezza dell'alfabeto. Nell'articolo si osserva che il comportamento di $\mathbb{E}[R_n]$ dipende dal tipo di correlazione tra n e k. È infatti semplice osservare che, per un k fissato, al tendere di n all'infinito $\mathbb{E}[R_n]$ converge a k. Caso più complesso si presenta quando $k = k(n)$ tende all'infinito insieme ad n.
I risultati ottenuti descrivono il comportamento del valore atteso sotto tre diverse ipotesi di dipendenza asintotica tra n e $k\sqrt{k}$.

Proposizione. *Sono verificate le sequenti affermazioni:*

- $n = \mathrm{o}(k\sqrt{k}) \to \lim\limits_{n\to\infty} \frac{\mathbb{E}[R_n]\cdot\sqrt{k(n)}}{n} = 2$
- $n = \frac{1}{2}\rho k\sqrt{k}$ per $\rho > 0 \to \liminf\limits_{n\to\infty} \frac{\mathbb{E}[R_n]}{k(n)} \geq 1 - e^{-\rho}$
- $n = (\frac{1}{2} + \xi)k\sqrt{k}\log k$ per un $\xi > 0 \to \lim\limits_{n\to\infty} \frac{\mathbb{E}[R_n]}{k(n)} = 1$

Queste proprietà sono vantaggiose durante gli studi comparativi, in quanto permettono di valutare la bontà di un algoritmo confrontando i valori di funzione obiettivo ottenuti, con il valore ottimo atteso $\mathbb{E}[R_n]$. Utilizzare questi risultati conviene soprattutto se n è grande, dove la proprietà diventa più affidabile e il calcolo del problema più oneroso.

Detta $\mathcal{H}$ un'euristica che risolve `RFLCS`. Una valutazione preliminare di $\mathcal{H}$, su un'istanza (x, y), si può fare controllando che

$$k \cdot (1 - e^{-\rho}) \leq |\mathcal{H}(x, y)| \leq k$$

dove la parte sinistra costituisce un limite inferiore che si verifica per n sufficientemente grande e $\rho > 0$ (si veda la seconda proprietà), mentre la parte destra è un limite superiore sempre vero per soluzioni ammissibili.

Ad esempio, facendo riferimento al data-set utilizzato durante le sperimentazioni (sezione 4.3), si supponga di valutare un'istanza tale che $n = 4096$ e $k = \frac{n}{8} = 512$. Si calcola $\rho = \frac{2n}{k\sqrt{k}} = 0.71$ e si valuta $260 \leq |\mathcal{H}(x, y)| \leq 512$, per un'euristica $\mathcal{H}$ e una generica istanza (x, y).

2.3 Soluzioni Proposte in Letteratura

Ad oggi non esiste in letteratura un algoritmo di approssimazione con rapporto di prestazione costante per il problema `RFLCS`.

I risultati teorici e sperimentali hanno evidenziato la difficoltà del problema in esame, giustificando le numerose proposte di algoritmi per il calcolo di soluzioni sub-ottime. Troviamo svariati approcci metodologici, quali: algoritmi randomizzati, branch-and-cut, programmazione dinamica e meta-euristiche. Si ritiene che il problema sia ancora ampiamente da esplorare sotto quest'ottica, alla luce dei limiti che caratterizzano ognuna delle proposte precedentemente elencate. Si procede con una presentazione dei metodi.

2.3.1 Algoritmi di Approssimazione

In [1] sono proposti 3 algoritmi di approssimazione con rapporto di prestazione dipendente da una variabile.

In prima analisi si trascrivono qui gli algoritmi. La descrizione fa uso di due costrutti quali, $f(s, \sigma)$, il cui valore indica la frequenza di σ nella stringa s, e $m_\sigma = \texttt{min}\{f(x, \sigma), f(y, \sigma)\}$ per un'istanza (x, y).

- Algoritmo $\mathcal{A}1$ - data un'istanza (x, y), calcolarne la Longest Common Subsequence, rimuovendo dalla soluzione, per ogni lettera di frequenza k, $k-1$ occorrenze arbitrarie.

- Algoritmo $\mathcal{A}2$ - data un'istanza (x, y), per ogni lettera σ tale per cui $f(x, \sigma) \leq f(y, \sigma)$, si eliminino $f(x, \sigma) - 1$ occorrenze random in x, si pro-

ceda analogamente su y per il caso $f(x,\sigma) > f(y,\sigma)$. Si calcoli la longest common subsequence per la sotto-istanza appena ottenuta.

- Algoritmo $\mathcal{A}3$ - si procede in maniera simile a quanto fatto in $\mathcal{A}2$. Data un'istanza (x,y), si sceglie un valore $r \in [0,1]$ tramite generazione uniforme random. Per ogni lettera σ tale per cui $f(x,\sigma) \leq f(y,\sigma)$ si sceglie quale occorrenza mantenere in x con un criterio deterministico basato sul valore di r, ad esempio, si scompone l'intervallo $[0,1]$ in $n(x,\sigma)$ sotto-intervalli di uguale grandezza e si verifica a quale intervallo appartiene r, così facendo è univocamente determinabile la scelta dell'occorrenza. Procedimento analogo va fatto su y per il caso $f(x,\sigma) > f(y,\sigma)$. Si calcola la longest common subsequence per la sotto-istanza appena ottenuta.

Mentre $\mathcal{A}1$ procede in maniera deterministica, gli algoritmi $\mathcal{A}2$ e $\mathcal{A}3$ sono algoritmi probabilistici. Il grado di approssimazione sarà quindi un valore ρ tale per cui $\mathbb{E}[|\mathcal{A}(x,y)|] \geq \frac{1}{\rho} \cdot \texttt{OPT}(\texttt{RFLCS}(x,y))$, con $\mathcal{A}$ algoritmo probabilistico.
Sia $m = \max_{\sigma \in \Sigma} m_\sigma$, è possibile dimostrare che gli algoritmi $\mathcal{A}1$, $\mathcal{A}2$ e $\mathcal{A}3$ sono delle m-approssimazioni per il problema RFLCS.

Teorema. *$\mathcal{A}1$ è una m-approssimazione per RFLCS(x,y).*

Dimostrazione. É chiaro che, calcolando una LCS, la soluzione ottenuta ha per ogni lettera un numero di occorrenze limitato superiormente da m. L'algoritmo sarà quindi una m-approssimazione per $\texttt{LCS}(x,y)$. Siccome $\texttt{OPT}(\texttt{RFLCS}(x,y)) \leq \texttt{OPT}(\texttt{LCS}(x,y))$, segue la tesi. □

Teorema. *[1] $\mathcal{A}2$ è una m-approssimazione per RFLCS(x,y).*

Dimostrazione. Per una generica sottosequenza z' di una sequenza z, si denota con $I(z',z)$ un insieme arbitrario di indici di z che corrisponde ad un'occorrenza di z' in z. Ad esempio, sia $z' = abc$ e $z = cacbbac$, l'insieme $\{2,4,7\}$ identifica un'occorrenza di z' in z, mentre l'insieme $\{2,3,5\}$ non trova corrispondenza.

Fissata un'istanza (x, y), sia w una soluzione ottima per l'istanza. Si ricordi che x' e y' sono le sequenze ottenute tramite la fase di eliminazione dell'algoritmo, inoltre $w' = \mathcal{A}2(x, y)$. Si vuole definire una variabile aleatoria L che identifica la lunghezza di una sottosequenza comune ad x' ed y' costituente un limite inferiore per $|w'|$.

Per ogni simbolo σ in w, sia Z_σ una variabile bernoulliana che assume valore 1 se e solo se (i) $m_\sigma(x, y) = f(x, \sigma)$ e l'indice in $I(w, x)$ che corrisponde a σ è in $I(x', x)$ (significa che la scelta random per σ è avvenuta su x); oppure (ii) $m_\sigma(x, y) = f(y, \sigma)$ e l'indice in $I(w, y)$ che corrisponde a σ è in $I(y', y)$.

Sia $Z = \sum Z_\sigma$, dove la sommatoria considera tutte i simboli occorrenti in w. Si osservi che la sottosequenza di w, ottenuta considerando i soli simboli σ tali che $Z_\sigma = 1$, corrisponde ad una sottosequenza sia di x' che di y'. Ciò implica che $Z \leq |w'|$ e che $\mathbb{E}[|w'|] \geq \mathbb{E}[Z]$. Per la linearità del valore atteso, $\mathbb{E}[Z] = \sum \mathbb{E}[Z_\sigma]$. Siccome Z_σ ha supporto binario, $\mathbb{E}[Z_\sigma] = \mathbb{P}(Z_\sigma = 1)$. La scelta casuale di σ avviene sempre in una sequenza che ha $m_\sigma(x, y) \leq m$ occorrenze di σ ed ogni scelta segue una distribuzione uniforme, quindi $\mathbb{P}(Z_\sigma = 1) \geq \frac{1}{m}$.

Si conclude che

$$\mathbb{E}[|w'|] \geq \mathbb{E}[Z] = \sum \mathbb{E}[Z_\sigma] \geq \frac{|w|}{m} = \frac{1}{m}\texttt{OPT}(\texttt{RFLCS}(x, y))$$

e l'algoritmo $\mathcal{A}2$ è una m-approssimazione per $\texttt{RFLCS}(x, y)$. □

Teorema. *$\mathcal{A}3$ è una m-approssimazione per $\texttt{RFLCS}(x, y)$.*

Dimostrazione. La dimostrazione procede in modo del tutto analogo a quanto fatto per l'algoritmo $\mathcal{A}2$. É infatti sufficiente notare che, anche con il criterio di scelta adottato da $\mathcal{A}3$, risulta sempre $\mathbb{P}(Z_\sigma = 1) \geq \frac{1}{m}$ per ogni σ che occorre in w soluzione ottima per il problema. □

Per completezza si fornisce una famiglia di istanze per la quale l'algoritmo $\mathcal{A}3$ esibisce le soluzioni sub-ottime peggiori possibili. Più formalmente, si definisce un

particolare insieme di istanze $\mathcal{F}$ tale che $\forall(x,y) \in \mathcal{F}, |\mathcal{A}3(x,y)| = \frac{\texttt{OPT}(\texttt{RFLCS}(x,y))}{m}$, con $m = \max_{\sigma \in \Sigma} m_\sigma$. A seguire si fa uso della notazione standard v^n per indicare la concatenazione consecutiva di $n \in \mathbb{N}$ ripetizioni della stringa v. Sia dunque $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$ l'alfabeto. Detto $x = \sigma_1\sigma_2\cdots\sigma_k$, le istanze sono della forma $(x^m, \sigma_k^{m+m_k}\sigma_{k-1}^{m+m_{k-1}}\cdots\sigma_1^{m+m_1})$, con $m, m_1, m_2, \ldots, m_k \in \mathbb{N}^+$. Resta da osservare il comportamento dell'algoritmo, racchiuso nella dimostrazione del prossimo teorema.

Teorema. $\mathcal{A}3$ *ha rapporto di prestazione limitato inferiormente da* m*.*

Dimostrazione. L'algoritmo inizia con la generazione di un valore $r \in [0,1]$ e, per ogni $\sigma \in \Sigma$, la scelta di quale elemento conservare ricade sempre tra le occorrenze di x^m. Qualunque sia il valore di r estratto, la sotto-istanza risultante dal processo di eliminazione è univoca, ossia $(x, \sigma_k^{m+m_k}\sigma_{k-1}^{m+m_{k-1}}\cdots\sigma_1^{m+m_1})$. Siccome i simboli di Σ appaiono, nella prima stringa, in ordine dal primo simbolo al k-esimo, mentre nella seconda stringa appaiono in ordine inverso, dal k-esimo simbolo al primo, l'istanza è formata da sole coppie in stato di cross e quindi le soluzioni possibili sono tutte e sole quelle formate da un solo simbolo $\sigma_i \in \Sigma$, mentre la soluzione ottima è una qualsiasi sequenza $\sigma_{i_m}\sigma_{i_{m-1}}\cdots\sigma_{i_1}$, tale che $i_m > i_{m-1} > \cdots > i_1$. Per comprendere la veridicità di questa affermazione si osservi che, considerando una qualsiasi sotto-sequenza x di x^m, si è già visto che una soluzione ammissibile avrà al più un elemento di x e, siccome la sequenza si ripete esattamente m volte, il valore di funzione obiettivo ottimo è pari ad m, mentre $|\mathcal{A}3(x,y)| = |\sigma_i| = 1$. □

Si conclude con un esempio: fissato $\Sigma = \{a, b, c\}$, un'istanza che rispetti la forma appena definita è $(abcabc, cccbbbaaa)$, l'algoritmo applicherà il calcolo di una `LCS` a $(abc, cccbbbaaa)$ ritornando a, b o c, mentre le soluzioni ottime sono cb, ca e ba.

Un ulteriore algoritmo di approssimazione con criteri deterministico è proposto in [15]:

- Algoritmo $\mathcal{A}4$ - data un'istanza (x, y), rimuovere, sia da x che da y, tutti i simboli σ tali che $m_\sigma > \sqrt{\texttt{min}(l_x, l_y)}$, con l_x ed l_y lunghezze di x ed y, rispettivamente. Applicare l'algoritmo $\mathcal{A}1$ alla sotto-istanza ottenuta dall'operazione precedente.

L'algoritmo, che di fatto è una semplice estensione di $\mathcal{A}1$, risulta interessante a livello teorico come dimostra il seguente

Teorema. *[15]* $\mathcal{A}4$ *è una* $2\sqrt{\texttt{min}(l_x, l_y)}$*-approssimazione per* $\texttt{RFLCS}(x, y)$*.*

Dimostrazione. Il numero di simboli σ rimossi dall'algoritmo è al più $\sqrt{\texttt{min}(l_x, l_y)}$. Per assurdo, si supponga di rimuoverne di più, ma ogni simbolo ha almeno $\sqrt{\texttt{min}(l_x, l_y)}$ occorrenze, il numero di eliminazioni eccede la dimensione dell'istanza stessa, il che è contraddittorio. Quindi risulta vero che $\texttt{OPT}(\texttt{RFLCS}(x, y)) \leq \sqrt{\texttt{min}(l_x, l_y)} + \texttt{OPT}(\texttt{RFLCS}(x', y'))$, con (x', y') sotto-istanza generata dal primo processo di eliminazione.

Sia m' il grado di approssimazione di $\mathcal{A}1$ sulla sotto-istanza (x', y'). Chiaramente $m' \leq \sqrt{\texttt{min}(l_x, l_y)}$ e l'algoritmo $\mathcal{A}1$ fornisce un rapporto di prestazione non superiore a $\sqrt{\texttt{min}(l_x, l_y)}$ per (x', y'). Si conclude che

$$|\mathcal{A}4(x, y)| \geq \frac{\texttt{OPT}(\texttt{RFLCS}(x', y'))}{\sqrt{\texttt{min}(l_x, l_y)}} \geq$$

$$\geq \frac{\sqrt{\texttt{min}(l_x, l_y)} + \texttt{OPT}(\texttt{RFLCS}(x', y'))}{2\sqrt{\texttt{min}(l_x, l_y)}} \geq \frac{\texttt{OPT}(\texttt{RFLCS}(x, y))}{2\sqrt{\texttt{min}(l_x, l_y)}}$$

e $\mathcal{A}4$ è una $2\sqrt{\texttt{min}(l_x, l_y)}$-approssimazione per $\texttt{RFLCS}(x, y)$. □

I limiti superiori ricavati con le precedenti dimostrazioni, oltre che essere variabili, sono piuttosto blandi, fatta eccezione per $\mathcal{A}3$. Si ritiene che, a riguardo, il margine di raffinamento è ampio, in particolare nel caso dell'algoritmo randomizzato $\mathcal{A}2$, dove ci si è limitati a maggiorare il valore atteso della funzione obiettivo $|\cdot|$ sulla soluzione approssimata.

2.3.2 Euristica con Programmazione Dinamica

Dall'articolo [15] arriva un'euristica che fa uso della programmazione dinamica. Si riporta qui la funzione di ricorsione implementata dall'algoritmo.

Date due stringhe $x = (x_1, x_2, \ldots, x_n)$ e $y = (y_1, y_2, \ldots, y_m)$ definite in un alfabeto Σ. Si denota con $s(i : j)$ la sotto-stringa di una sequenza s che inizia da s_i e finisce con s_j.

Sia $\mathcal{S}(i, j)$ il sotto-insieme di Σ che caratterizza la soluzione per una sotto-istanza $(x(1 : i), y(1 : j))$. La sua definizione ricorsiva sarà

$$\mathcal{S}(i,j) = \begin{cases} \emptyset & \text{se } i = 0 \vee j = 0 \\ \mathcal{S}(i-1, j-1) \cup \{x_i\} & \text{se } x_i = y_j \wedge x_i \notin \mathcal{S}(i-1, j-1) \\ \mathcal{S}(i-1, j) & \text{se } |\mathcal{S}(i-1, j)| \geq |\mathcal{S}(i, j-1)| \\ \mathcal{S}(i, j-1) & \text{altrimenti} \end{cases} \tag{2.1}$$

Assumendo che $\mathcal{S}$ sia un insieme ordinato, $\mathcal{S}(n, m)$ sarà una soluzione repetition-free per l'istanza (x, y).

Nello stesso articolo si propone un'espansione di quest'algoritmo. Analizzando l'euristica precedente si è cercato di superare le principali limitazioni. Nella nuova euristica, denominata Top-k, una cella della matrice, anziché memorizzare un'insieme, consisterà in un insieme di insiemi.

	a	b	d	c	b	d
b		b	b	b	b	b
d		b	b,bd	b,bd	b,bd	b,bd
a	**a**	**a**,b	**a**,bd	a,bd	a,bd	a,bd
c	a	a,b	a,bd	**ac**,bdc	ac,bdc	ac,bdc
b	a	a,b	ab,bd	ac,bdc	**acb**,bdc	acb,bdc
d	a	a,ab	ab,abd	abd,bdc	acb,abd	acb,**acbd**

Tabella 2.1: matrice ottenuta dall'esecuzione dell'euristica Top-2 sull'istanza (*abdcbd*, *bdacbd*). La soluzione ricostruita, segnata in grasseto, è ottima.

Naturalmente conservare ogni possibile combinazione renderebbe l'euristica

intrattabile, ci si limita quindi a k insiemi. É chiaro che occorre definire un criterio di selezione. Trattasi degli insiemi la cui cardinalità dell'intersezione è minima. L'articolo procede con una fase di sperimentazione, in cui si è adottata un'euristica Top-20 che ha mostrato un ottimo rapporto tra tempi di calcolo e bontà della soluzione.

Ad ogni modo il metodo risulta piuttosto limitato dal punto di vista dello space-consuming, che cresce rapidamente all'aumentare della precisione richiesta.

2.3.3 Euristica Branch and Cut

In [18] si propone l'applicazione di un *Branch-and-Cut* al problema `RFLCS` con una nuova formulazione matematica che considera nei vincoli un costrutto detto **extended star**. Detto Z un'insieme di variabili di decisione definito analogamente a quanto fatto in 2.1. Una extended star è un qualsiasi sotto-insieme $Z' \subseteq Z$ tale che ogni coppia in $Z_{\sigma_1} \cap Z'$ è in stato di **cross** con le coppie in $Z_{\sigma_2} \cap Z'$ - si supponga $\sigma_1 \neq \sigma_2$.

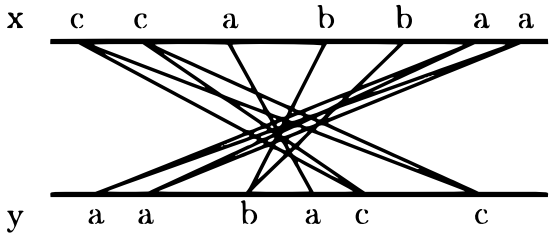


Figura 2.2: una extended star. Gli archi che non creano cross corrispondono alla stessa lettera, rispettando la definizione.

Una extended star massimale è una extended star alla quale non è possibile aggiungere altri elementi senza che perda la proprietà precedentemente definita.

Segue la formulazione matematica:

$$\begin{aligned} \max \quad & \sum_{z_{i,j} \in Z} z_{i,j} \\ \text{s.a} \quad & \sum_{z_{i,j} \in Z'} z_{i,j} \leq 1, \ \forall Z' \subseteq Z \text{ extended star massimale} \\ & z_{i,j} \in \{0,1\}, \ \forall z_{i,j} \in Z \end{aligned}$$

Il Branch-and-Cut proposto basa il piano di taglio sulla ricerca di una extended star massimale Z' che violi il vincolo di disuguaglianza. Operare tramite extended star risulta utile per via di una proprietà di cui gode il costrutto, che rende le operazioni di controllo dei vincoli molto efficienti. Se, a partire da un'istanza (x, y), si inverte una delle due stringhe , ad esempio x, si ottiene un'istanza (x^r, y) dove una coppia $(x^r_{|x|-i+1}, y_j)$ corrisponde alla coppia (x_i, y_j). Ogni extended star di (x, y) diventa, in (x^r, y), un *extended planar matching*. Per la definizione di un extended planar matching si definisce a sua volta un *blocco*, ossia un insieme di coppie dello stesso simbolo. Un extended planar matching è un insieme di blocchi che non provoca alcun cross - figura 2.3.

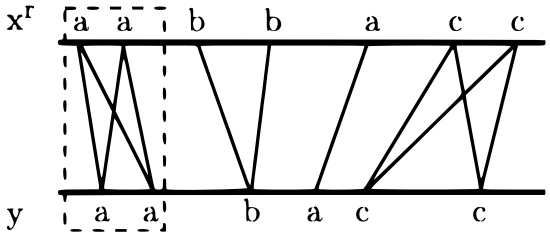


Figura 2.3: un extended planar matching. L'insieme di coppie all'interno della cornice tratteggiata è un blocco.

Cercare un'extended planar matching massimale risulta un metodo semplice ed equivalente alla ricerca di un extended star massimale.

L'algoritmo complessivo, oltre a sfruttare l'interessante proprietà strutturale appena descritta, offre un approccio in grado di risolvere istanze di piccola e media taglia all'ottimo, considerando istanze con lunghezze che variano tra 32 e

1024. Ciò nonostante, l'algoritmo diventa inutilizzabile rapidamente e non è in grado di risolvere istanze grandi in un range temporale di 30 minuti.

2.3.4 Metaeuristiche

Negli ultimi anni si sono fatti ulteriori passi avanti mediante l'uso di tecniche meta-euristiche. Nel 2013 viene presentata un'implementazione di un Evolutionary Algorithm (EA) in [19]. Il framework procede secondo i seguenti step:

1. generare la popolazione iniziale di individui in modo casuale
2. valutare l'idoneità di ciascun individuo in quella popolazione
3. ripetere i seguenti passaggi finché non si verifica il criterio di stop:
4. (a) selezionare gli individui più adatti per la riproduzione
   (b) creare nuovi individui attraverso operazioni di crossover e mutazione
   (c) valutare l'idoneità individuale di nuovi individui
   (d) sostituire la popolazione meno adatta con nuovi individui.

Questa meta-euristica è successivamente oggetto di comparazione con altre 3 applicazioni di meta-euristiche proposte in [20]. Gli algoritmi progettati sono frutto di un lavoro di ricerca per l'ottimizzazione del problema LCS. É in fatti possibile leggere tutti gli sviluppi nel seguente libro [7]. Le ultime proposte pubblicate sono dunque:

- Beam-ACO - una meta-euristica ibrida che fa uso di una componente di ricerca nota come Beam Search nel contesto di un algoritmo di tipo Ant Colony Optimization.
- CMSA - una meta-euristica ibrida che combina elementi euristici con l'applicazione di un generico risolutore per problemi di Programmazione Lineare Intera.

- Hyb-CMSA - un'evoluzione del CMSA precendente, tramite introduzione di una componente di ricerca, utilizzata nel Beam-ACO, che viene qui adottatta per la costruzione di una soluzione iniziale.

La comparazione dei primi due algoritmi non ha portato ad un distacco netto che permettesse di decretare quale fosse il migliore. Piuttosto si è osservato che il CMSA è migliore per istanze di taglia media e piccola, mentre Beam-ACO risulta più robusto all'aumento di dimensione delle istanze. Questi esperimenti hanno portato alla definizione di Hyb-CMSA, che ha portato a dei risultati sperimentali superiori alla controparte. Hyb-CMSA è da considerarsi, quindi, l'attuale stato dell'arte.
Segue una descrizione di Beam-ACO e CMSA framework.

Beam-ACO

Ant Colony Optimization (ACO) e Beam Search (BS) hanno la caratteristica comune di essere basati sull'idea di costruire le soluzioni candidate passo dopo passo. Tuttavia, i modi in cui i due metodi esplorano lo spazio di ricerca sono abbastanza diversi. Gli algoritmi BS sono guidati da due diverse componenti, ossia (i) una funzione di valutazione che viene utilizzata per ponderare le diverse possibilità di estendere una soluzione parziale, e (ii) il limite inferiore che viene utilizzato per limitare il numero di soluzioni parziali ad ogni passo. La politica utilizzata negli algoritmi BS per estendere soluzioni parziali è solitamente deterministica. Al contrario, gli algoritmi ACO effettuano un'esplorazione probabilistica, utilizzando l'esperienza passata al fine di trovare buone aree dello spazio di ricerca. In altre parole, il processo di ricerca di ACO è adattivo.
Ci si aspetta un beneficio dalla combinazione di questi due modi di esplorare uno spazio di ricerca. La base algoritmica di questa combinazione sarà il framework di ACO. Tuttavia, si sostituisce il meccanismo di costruzione della soluzione degli algoritmi ACO standard con un meccanismo di costruzione della soluzione tramite BS probabilistica.

CMSA

Data un'istanza di un problema $\mathcal{I}$ ad un problema generico $\mathcal{P}$, C rappresenta l'insieme di tutte le possibili componenti di cui sono composte le soluzioni di $\mathcal{I}$. C è d'ora in

poi chiamato set completo di componenti. In generale ogni combinazione di una variabile con uno dei suoi valori è una possibile componente. Inoltre, una soluzione S sarà un sotto-insieme $S \subseteq C$. Infine, un dato $C' \subseteq C$ contiene delle componenti che appartengono a una sotto-istanza di $\mathcal{I}$. Si consideri quindi, per semplicità, C' come una sotto-istanza dell'istanza C.

L'algoritmo Construct, Merge, Solve Adapt (CMSA) funziona nel seguente modo. Ad ogni iterazione, l'algoritmo genera un'istanza C', inizialmente vuota. Il primo passo di ogni iterazione consiste nel generare un numero di soluzioni ammissibili all'istanza del problema originale $\mathcal{I}$ in modo probabilistico. Nella seconda fase, le componenti coinvolte in queste soluzioni vengono aggiunte a C' e viene applicato un risolutore esatto per risolvere C'. Infine, il terzo passo consiste nell'adattare la sotto-istanza C' rimuovendo alcune componenti della soluzione guidate da un meccanismo di invecchiamento.

2.4 Conclusioni

Le meta-euristiche presentate costituiscono lo stato dell'arte in tema di ottimizzazione combinatoria per il problema `RFLCS`. Le metodologie coinvolte si è visto essere limitate da alcune caratteristiche intrinseche dell'approccio stesso.

- Gli algoritmi randomizzati offrono dei primi interessanti risultati teorici, ma i valori di funzione obiettivo sono lontani dai valori ottimi, come dimostrano le comparazioni con altri algoritmi.
- L'euristica con programmazione dinamica permette di calcolare delle soluzioni sub-ottime competitive, ma a costo di un elevato consumo di spazio.
- L'euristica branch-and-cut offre un approccio interessante al problema, con esito soddisfaciente su istanze di piccola e media taglia dal punto di vista della funzione obiettivo. Tuttavia l'algoritmo è lento e i tempi di attesa diventano insostenibili rapidamente, al crescere della taglia.

- Le meta-euristiche costituiscono l'attuale stato dell'arte in quanto sono robuste al crescere della taglia dell'istanza. Va però detto che i tempi di attesa sono piuttosto onerosi.

Come conseguenza di questa indagine, si ritiene opportuno continuare ad operare tramite l'uso di tecniche di ottimizzazione combinatoria con l'obiettivo di trovare soluzioni che esplorino aspetti interessanti a livello teorico e/o applicativo.

Capitolo 3

Progettazione delle Meta-euristiche

La classe delle euristiche è diventata una famiglia molto popolare di metodi di soluzione per problemi di ottimizzazione perché sono in grado di trovare soluzioni accettabili in un ragionevole lasso di tempo.

Negli ultimi decenni, i progressi algoritmici e i miglioramenti hardware e software hanno fornito un ambiente eccellente su cui costruire sistemi di supporto alle decisioni tramite criteri euristici.

Con tecniche euristiche ci si riferisce ad approcci basati sull'esperienza per la risoluzione di problemi. Fanno parte di questa categoria le *meta-euristiche*, delle metodologie di risoluzione che si astraggono dal problema specifico, delineando dei procedimenti di ricerca all'interno di un generico spazio delle soluzioni.

Questo capitolo è dedicato alla presentazione delle meta-euristiche adottate durante l'attività di tesi. Ad ogni algoritmo è dedicata una sezione, nella quale viene descritto il processo metodologico tramite una prima descrizione del framework scelto, seguita dall'esposizione delle scelte implementative.

Oltre la descrizione delle meta-euristiche, in questo capitolo si trova una sezione dedicata alle strutture dati utilizzate, seguita da una descrizione tecnica per l'implementazione di un modulo responsabile di tutte le operazioni di campionamento casuale. Infine si

riporta la pseudo-codifica scelta per la risoluzione del LCS problem, calcolo ricorrente all'interno degli algoritmi implementati.

3.1 Greedy Randomized Adaptive Search Procedure

Descrizione Framework

Un algoritmo di tipo GRASP è classificabile come algoritmo iterativo multi-start, ovvero un framework che prevede la costruzione di diverse soluzioni interindipendenti. Alla base di un'approccio multi-start vi è l'obiettivo di offrire un'esplorazione diversificata dello spazio di ricerca per evitare di bloccarsi in ottimi locali. La meta-euristica GRASP, introdotta in [21], è una delle più famose tecniche multi-start proposte in letteratura.

L'algoritmo inizia impostando come soluzione migliore trovata, la soluzione vuota. Dopodichè l'algoritmo entra nella componente ciclica. Internamente ad una generica iterazione avviene la costruzione di una soluzione ammissibile, a partire dalla quale, si effettua una fase di ricerca locale parametrizzata secondo una data funzione *neighborhood* $\mathcal{N}$, ovvero una funzione che vuole come unico parametro una soluzione ammissibile al problema e che restituisce un insieme di soluzioni ammissibili ad essa "vicine". L'algoritmo continua ad alternare queste fasi di costruzione e ricerca finchè non si verifica il criterio di arresto.

Sia dunque $\mathcal{I}$ un'istanza di un problema di ottimizzazione, s un seed per la generazione pseudo-casuale di elementi, p la funzione profitto da massimizzare e g una funzione greedy. L'algoritmo 1 riporta la pseudo-codifica del pattern GRASP. La fase di costruzione è un processo, iterativo ed adattivo, guidato da una funzione di valutazione greedy. É iterativo in quanto la soluzione iniziale è costruita considerando un elemento alla volta. È greedy perchè la scelta di ogni elemento e è guidata da una funzione che stima il beneficio che porterebbe l'inserimento di e nella soluzione parzialmente costruita. Infine è adattivo perchè la scelta in una singola iterazione è in funzione delle scelte

Algorithm 1 GREEDY RANDOMIZED SEARCH PROCEDURE

1: **procedure** GRASP($\mathcal{I}, \mathcal{N}, s, p, g$)
2: $x^* \leftarrow \emptyset$
3: **repeat**
4: $x \leftarrow$ CONSTRUCT($\mathcal{I}, g, s \mid \theta$)
5: **if** $\neg$FEASIBLE(x) **then** $x \leftarrow$ REPAIR(x)
6: $x \leftarrow$ LOCALSEARCH($x, \mathcal{N}, p$)
7: **if** $p(x) > p(x^*)$ **then** $x^* \leftarrow x$
8: **until** termination condition met
9: **return** x^*

prese alle iterazioni precedentemenenti.

La funzione di costruzione alla riga 4 è parametrizzata per un vettore generico θ. Esistono infatti diversi pattern di costruzione proposte in letteratura. Si è deciso di riportare la prima versione creata, seguita da un'estensione che introduce una componente di *biasing randomization* ([22], [23]). L'algoritmo 2 inizia ordinando gli elementi dell'istanza

Algorithm 2 CONSTRUCTION WITH RESTRICTED CANDIDATE LIST

1: **procedure** CONSTRUCT($\mathcal{I}, g, s \mid \alpha$)
2: $x \leftarrow \emptyset$
3: $\mathcal{L} \leftarrow$ SORTEDLIST($\mathcal{I}, g$)
4: **repeat**
5: $\mathcal{L}_r \leftarrow$ RESTRICTEDCANDIDATELIST($\mathcal{L}, \alpha$)
6: $e \leftarrow$ SELECT($\mathcal{L}_r, s$)
7: $x \leftarrow x \cup \{e\}$
8: ADAPT($\mathcal{L}, g, e$)
9: **until** COMPLETE(x)
10: **return** x

$\mathcal{I}$ secondo la funzione greedy g. Successivamente inizia la costruzione iterativa, che termina quando si verifica il criterio di stop (riga 9). All'interno del ciclo si genera la *Restricted Candidate List*, una lista ristretta dei migliori elementi tra quelli candidati ad entrare in soluzione. Quanto debba essere grande $\mathcal{L}_r$ non è noto a priori ed è quindi una funzione del parametro α. A questo punto si effettua una scelta casuale uniforme di un elemento e all'interno di $\mathcal{L}_r$ e si aggiunge in soluzione. L'iterazione si conclude con l'aggiornamento dell'ordinamento degli elementi, coerentemente con la scelta fatta (riga 8). La pseudocodifca descritta dall'algoritmo 3 si differenzia dalla prima versione per la selezione di un elemento. Mentre prima avveniva una scelta secondo

Algorithm 3 CONSTRUCTION WITH BIASED RANDOMIZATION

```
1: procedure CONSTRUCT(I, g, s | D)
2:     x ← ∅
3:     L ← SORTEDLIST(I, g)
4:     repeat
5:         e ← SELECT(L, s, D)
6:         x ← x ∪ {e}
7:         L ← L \ {e}
8:         ADAPT(L, g, e)
9:     until COMPLETE(x)
10:    return x
```

una generazione uniforme random in una lista di candidati ristretta, in questo caso si seleziona un elemento tra tutti i candidati in accordo con un parametro $\mathcal{D}$ che specifica la distribuzione di probabilità.

Implementazione

L'algoritmo qui presentato è un'implementazione del framework con costruzione di tipo *biased randomized*.

La soluzione incombente viene inizializzata con la stringa vuota. Successivamente inizia la fase iterativa in cui si calcolano delle soluzioni. Come criterio di arresto per questa fase si usa un budget massimo utilizzabile. Il budget scelto è un tempo massimo in cui l'algoritmo può lavorare.

Il calcolo di una soluzione è fatta di due fasi. Una prima fase in cui si costruisce una sotto-istanza di partenza caratterizzata da proprietà particolari ed una seconda fase che comprende la risoluzione all'ottimo di un problema di Longest Common Subsequence (polinomialmente risolvibile) per la sotto-istanza ottenuta al passo uno. Nello specifico, la sotto-istanza gode della seguente proprietà: per ogni lettera dell'alfabeto, il numero delle rispettive occorrenze è al più 1 per una delle due stringhe.

La costruzione della sotto-istanza avviene in modo greedy randomizzato. Il procedimento parte dall'istanza originale e, per ogni lettera dell'alfabeto σ, effettua una selezione greedy randomizzata di una occorrenza di σ. Detta (x, y) un istanza del problema, si supponga ad esempio che viene scelto l'elemento $x_i = \sigma$. In questo caso verranno eliminate da x tutte le occorrenze di σ ad eccezione di x_i. Si procede analogamente se

l'elemento scelto è in y. A procedimento effettuato per tutte le lettere, la nuova sotto-istanza godrà della proprietà precedentemente descritta. Risulta inoltre evidente che il calcolo di una Longest Common Subsequence porterà ad una soluzione ammissibile per RFLCS. Di conseguenza a partire dalla sotto-istanza costruita nella prima fase, si utilizzerà un algoritmo di LCS per ottenere una soluzione nella seconda fase.
A partire da questa soluzione inizia la fase di ricerca locale. La *neighborhood* considerata prova ad inserire nella soluzione incombente, cioè nella sotto-sequenza, tutti i caratteri dell'alfabeto che ancora non vi compaiono, tentando di frapporli tra ciascuna coppia di caratteri già inserita. La fase di ricerca locale si interrompe quando non è più possibile migliorare la soluzione.

3.1.1 Criterio Greedy

Sia (x, y) un'istanza per il problema RFLCS e l_s la lunghezza di una stringa s. Si definisce una funzione distanza $\delta : \mathbb{N} \times \mathbb{N} \to \mathbb{R}^+$, per una coppia di elementi (x_i, y_j), come

$$\delta(i,j) = \begin{cases} |i - j\frac{l_x}{l_y}| & \text{se } l_x > l_y \\ |j - i\frac{l_y}{l_x}| & \text{altrimenti.} \end{cases} \tag{3.1}$$

Siano I_j, J_i due insiemi di pedici così definiti

$$J_i = \{j \mid z_{i,j} \in Z_{x_i}\}$$

$$I_j = \{i \mid z_{i,j} \in Z_{y_j}\}$$

Il criterio greedy peserà una lettera di una stringhe in base alla distanza media dalle occorrenze della stessa nell'altra stringa, matematicamente si ha

$$g(x_i) = \frac{1}{|J_i|} \sum_{j \in J_i} \delta(i,j)$$

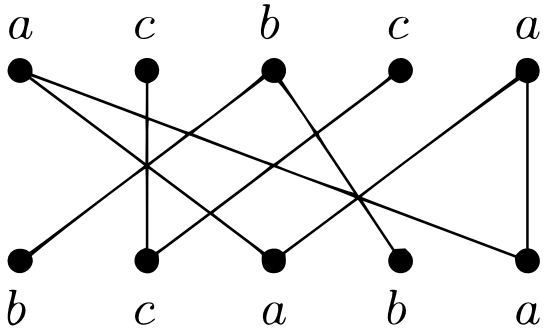
$$g(y_j) = \frac{1}{|I_j|} \sum_{i \in I_j} \delta(i,j)$$

Qualora l'insieme J_i, rispettivamente I_j, risulti vuoto allora $g(x_i) = \infty$, rispettivamente $g(y_j)$.

Si noti che la funzione distanza δ prevede il caso generico di istanze come coppie di stringhe di lunghezza diversa. L'operazione tenta di essere robusta al crescere di $|l_x - l_y|$.

Il ragionamento da cui è emersa questa definizione è piuttosto intuitivo: se si trova un sotto-insieme $Z' \subseteq Z$ i cui elementi hanno una buona probabilità di causare un numero di cross basso, relativamente ad altri elementi di Z, ci si aspetta che il numero di conflitti all'interno di Z' sia basso. Di conseguenza un numero elevato di elementi può trovarsi all'interno di una stessa soluzione, portando ad un buon valore di funzione obiettivo.

Ma allora che caratteristiche deve avere una variabile $z_{i,j}$ perchè sia verosimile che il numero di cross che provoca sia basso. Si consideri il seguente esempio:



detta x la sequenza *acbca* ed y la sequenza *bcaba*, ci si focalizzi sulle coppie (x_1, y_5) e (x_5, y_5). Il primo caso è in stato di cross con 5 coppie, mentre il secondo non provoca nessun conflitto. Non a caso per (x_1, y_5) si ha che $\delta(1, 5) = 4$ che è anche il numero più grande possibile, per l'istanza che si sta considerando. Mentre $\delta(5, 5) = 0$ è la distanza tra gli elementi della seconda coppia, classificata come molto appetibile, come di fatto è, in quanto presente nella soluzione ottima *bca*, mentre la prima coppia è presente solo nella soluzione peggiore composta dalla sola a. Volendo associare un'grado di appetibilità ai signoli elementi è immediata la scelta di utilizzare la media sopra definita. Ad x si associa la sequenza di valori greedy $(3.0, 0.0, 1.5, 2.0, 1.0)$, mentre da y si ottiene $(2.0, 1.0, 2.0, 1.0, 2.0)$. Si osservi che i valori più bassi corrispondono ancora ad elementi presenti nelle soluzioni ottime e viceversa, i valori più bassi tendono a portare a soluzioni

peggiori.

Entrando nel merito della costruzione di una soluzione, utilizzata nel criterio greedy, si avrà per ogni lettera dell'alfabeto $\{a, b, c\}$ una sequenza ordinata delle occorrenze, rispetto ai valori greedy. Ossia:

a (x_5, y_3, y_5, x_1)

b (y_4, x_3, y_1)

c (x_2, y_2, x_4)

Se si applica il metodo di costruzione in maniera greedy deterministica si ottiene una sotto-istanza formata dalle stringhe *cba* e *caba*. Risolvendo il problema rilassato LCS, si ottiene la soluzione *cba*, ottima per il problema RFLCS, calcolata in tempo polinomiale sulla taglia dell'istanza. In particolare il calcolo dei valori greedy può essere effettuato in tempo lineare sulla lunghezza delle sequenze; il costo di ordinamento per una sequenza associata ad una lettera σ è $\mathcal{O}(f_\sigma \log f_\sigma)$ con $f_\sigma = f(x, \sigma) + f(y, \sigma)$, ordinare tutte le sequenze è quindi ancora polinomiale; la generazione di una sotto-istanza può essere fatto in $\mathcal{O}(|x| + |y|)$, se si procede per eliminazione degli elementi a partire dall'istanza originaria (3.6); infine il calcolo di una LCS avviene in $\mathcal{O}(|x| \cdot |y|)$.

3.2 Generate and Solve

Descrizione Framework

Introdotto in [24], il framework Generate and Solve appartiene alla classe delle meta-euristiche ibride [25], ossia degli algoritmi che combinano le componenti di diverse tecniche di ottimizzazione. Il GS è basato sulla possibilità che il problema da affrontare presenti delle caratteristiche tali per cui, per una data istanza è possibile individuare delle sotto-istanze ridotte, le cui soluzioni di buona qualità, risultano buone anche per l'istanza originale.

La procedura è composta da due componenti fondamentali, quali

- Generatore di Istanze Ridotte (GIR)

- Decodificatore di Istanze Ridotte (DIR)

Il GIR può essere implementato con una qualsiasi euristica in grado di generare una sotto-istanza $\mathcal{I}'$ a partire da un'istanza $\mathcal{I}$ da risolvere per un problema $\mathcal{P}$.
La componente DIR è un modulo che calcola una soluzione per un'istanza data. Solitamente la componente scelta è un risolutore esatto, ossia che garantisce la risoluzione all'ottimo della sotto-istanza passata in input. Non vi sono tuttavia imposizioni a riguardo e sono ammesse componenti di ogni tipo.
I due moduli sono interconnessi, nel senso che, così come il DIR dipende dall'istanza generata dal GIR, il GIR opererà in funzione di un feedback ottenuto dal DIR.
Il framework consisterà dunque di un main loop all'interno del quali si alternano i due moduli appena descritti.
Siccome gli autori non hanno presentato una pseudocodifica per il GS, se ne propone una qui, in cui si introduce la variabile θ, rappresentante il feedback ottenuto dal DIR in forma di vettore di parametri. Siano inoltre $\mathcal{I}$ un'istanza per un problema generico e p una funzione profitto da massimizzare.

Algorithm 4 GENERATE AND SOLVE FRAMEWORK

```
1: procedure GS(I, p, θ)
2:     x* ← ∅
3:     repeat
4:         I' ← GENERATE(I, theta)
5:         x ← SOLVER(I')
6:         UPDATE(θ, x)
7:         if p(x) > p(x*) then x* ← x
8:     until termination condition met
9:     return x*
```

Implementazione

A differenza di altri framework, in questo caso sono descritti esclusivamente pochi macro-passi, consentendo un alto grado di libertà metodologica.
L'algoritmo parte da una sotto-istanza vuota e ad ogni iterazione crea una sotto-istanza a partire dalla precedente.

Algorithm 5 GENERATE AND SOLVE

1: **procedure** GS($\mathcal{I} = (x, y), |\cdot|, \tau, \text{age}$)
2: $s^* \leftarrow \emptyset,\ \mathcal{I}' \leftarrow (\emptyset, \emptyset)$
3: **repeat**
4: $\mathcal{I}' \leftarrow$ INSTANCEGENERATOR($\mathcal{I}, \mathcal{I}', \tau, \text{age} \mid \mathcal{D}$)
5: $s \leftarrow$ TWOSTEPSOLVER($\mathcal{I}'$)
6: UPDATE(s, τ, age)
7: **if** $|s| > |s^*|$ **then** $s^* \leftarrow s$
8: **until** time limit met
9: **return** s^*

La generazione di una istanza parte dall'istanza $\mathcal{I}'$ costruita all'iterazione precedente, alla quale aggiunge nuovi elementi ed elimina quelli considerati troppo vecchi e controproducenti al calcolo di una soluzione di buona qualità. Il costo dell'eliminazione (righe 2 a 5) è lineare sulla taglia dell'istanza corrente $\mathcal{I}'$, mai più grande dell'istanza originale $\mathcal{I}$. L'inserimento di nuovi elementi è composto da un ciclo sull'alfabeto e suppone, senza perdita di generalità, che $|\Sigma| \leq \texttt{max}\{|x|, |y|\}$. Per ogni simbolo σ si genera un numero di occorrenze limitato superiormente da $f(\sigma, x) + f(\sigma, y)$, anche questa fase è quindi lineare, in questo caso sulla taglia di $\mathcal{I}$.

Algorithm 6 GENERATOR OF REDUCED INSTANCES

1: **procedure** INSTANCEGENERATOR($\mathcal{I}, \mathcal{I}', \tau, \text{age} \mid \mathcal{D}, A_{max}$)
2: **for all** $x'_i \in x'$ **do**
3: **if** $\text{age}_x(i) = A_{max}$ **then** $x' \leftarrow x' \setminus x'_i$
4: **for all** $y'_j \in y'$ **do**
5: **if** $\text{age}_y(j) = A_{max}$ **then** $y' \leftarrow y' \setminus y'_j$
6: **for all** $\sigma \in \Sigma$ **do**
7: $x' \leftarrow x' \cup$ GENERATE($\tau(\sigma), x, \mathcal{I}, \mathcal{D}$)
8: $y' \leftarrow y' \cup$ GENERATE($\tau(\sigma), y, \mathcal{I}, \mathcal{D}$)
9: **return** $\mathcal{I}'$

Si definiscono le seguenti:

- età massima A_{max} - se l'età associata ad un elemento di una stringa raggiunge questo valore, le variabili di decisione associate a quell'elemento vengono eliminate dalla sotto-istanza
- $\tau : \Sigma \to [0, 1]$ - per una lettera $\sigma \in \Sigma$ si ha che $\tau(\sigma) \cdot f(v, \sigma)$ è il numero di generazioni di occorrenze di v di σ (si ricorda che $f(v, \sigma)$ è la frequenza di σ per

una stringa v)

- distribuzione di probabilità $\mathcal{D}$ - la generazione degli elementi da inserire in soluzione avviene in accordo alla distribuzione $\mathcal{D}$, calcolato a partire dal criterio greedy descritto in 3.1.1.

L'implementazione del DIR consiste in un risolutore a due passi deterministici. Al primo passo viene calcolata una Longest Common Subsequence ($\texttt{LCS} \in \texttt{P}$), dopodiché a partire dalla soluzione calcolata si eliminano le ripetizioni, operazione effettuabile in tempo lineare sulla soluzione, mai più grande di $\texttt{min}\{|x|, |y|\}$, per un'istanza (x, y). Invece di mantenere una coppia a caso, si sceglie di far sopravvivere la coppia che minimizza la funzione distanza δ definita in 3.1.1 e calcolabile in $\mathcal{O}(1)$.
Infine avviene l'aggiornamento dei parametri di feedback (riga 6). Per ogni lettera σ si valuta se ridurre il valore $tau(\sigma)$ o aumentarlo in base al numero di ripetizioni calcolate nel primo passo del DIR, conteggio effettuabile in tempo lineare sulla `LCS` calcolata. Per ogni occorrenza presente in $\mathcal{I}'$ che non è sopravvissuta al calcolo viene incrementata l'età e se sfora l'età massima, si elimina dall'istanza, in entrambi i casi la complessità è $\mathcal{O}(1)$.

3.3 Iterated Local Search

Descrizione Framework

Iterated Local Search è una meta-euristica che accorpa una componente di miglioramento euristica all'interno di un processo iterativo che genera una catena di soluzioni. Il processo iterativo consiste in una perturbazione della soluzione corrente, così facendo si approda ad una nuova soluzione che sarà il punto di partenza per la fase di miglioramento.
Il metodo si è evoluto nel tempo in diversi modi e sono state prese in considerazione una serie di nuove varianti e generalizzazioni. Infatti, una prima generalizzazione afferma che l'euristica di miglioramento non deve essere necessariamente un algoritmo di miglioramento iterativo, ma può essere qualsiasi metodo che prende una soluzione

come input e restituisce una soluzione probabilmente migliorata.

Alcuni esempi applicativi sono reperibili nel libro [26].

A seguire si riporta la pseudocodifca del pattern.

Algorithm 7 ITERATED LOCAL SEARCH FRAMEWORK

```
1: procedure ILS(I, p)
2:     x ← BuildSolution(I)
3:     x* ← x
4:     repeat
5:         x ← Perturbation(I, x)
6:         x ← LocalSearch(I, x)
7:         if p(x) > p(x*) then x* ← x
8:     until termination condition met
9:     return x*
```

Implementazione

L'algoritmo implementato prevede una prima costruzione di una soluzione secondo il criterio gia utilizzato in 3.1. A seguito della costruzione inizia un ciclo in cui si susseguono delle ricerche locali su soluzioni ottenute tramite tecniche di diversificazione.

La ricerca locale consiste in una serie di tentativi di espansione della soluzione. In particolare si valutano tutti i simboli dell'alfabeto che non occorrono nella soluzione corrente e, per ognuno di essi, si tenta un inserimento in soluzione. Questo tentativo comporta la ricerca di una locazione tra 2 lettere della soluzione in cui è possibile inserire il simbolo candidato. L'inserimento va a buon fine se è possibile individuare una sottosequenza nell'istanza originale che corrisponda alla nuova soluzione che si sta cercando di costruire. Ciò può essere fatto in un tempo che è, nel caso peggiore, lineare sulla taglia dell'istanza, ma può essere ottimizzato tramite l'esplorazione dell'istanza codificata con una *mappa delle posizioni* (si veda sezione strutture dati, 3.6).

La fase di diversificazione effettua delle operazioni di distruzione parziale della soluzione, escludendo alcuni elementi. La scelta di quali eliminare è eseguita in maniera del tutto casuale e con complessità lineare sul numero di elementi da eliminare. Quanti eliminarne è una funzione di un parametro che va determinato in fase di sperimentazione. La soluzione ottenuta dalla perturbazione viene data in input alla ricerca locale (riga

5). Questa componente è la stessa utilizzata per l'algoritmo GRASP e per i dettagli si rimanda alla lettura della sezione 3.1. Per la stesura della pseudocodifica 8 si utilizza $\mathcal{D}$ per una distribuzione di probabilità e α per il parametro per il calcolo del numero di elementi da eliminare.

Algorithm 8 ITERATED LOCAL SEARCH

```
1: procedure ILS(I = (x, y), |·|)
2:     s ← GreedyRandomBuild(I | D)
3:     s* ← s
4:     repeat
5:         s ← Destroy(s | α)                    ▷ Perturbation
6:         s ← Insert(s, I)                      ▷ Local Search
7:         if |s| > |s*| then s* ← s
8:     until time limit met
9:     return s*
```

3.4 Operazioni di Campionamento

Una componente dedicata è stata progettata per gestire tutte le operazioni che prevedono il campionamento pseudo-casuale. Questo modulo permette di generare dei campioni di una generica variabile aleatoria $\mathbb{X}_\sigma : \{0, 1, \ldots, f_\sigma - 1\} \to \mathbb{R}$, con f_σ frequenza della lettera σ per una data istanza. Data una generica distribuzione di probabilità $\mathcal{D}$ è possibile generare un elemento dallo spettro di $\mathbb{X}_\sigma$. Abbinando questa operazione alla lista dei candidati precedentemente descritta (3.6.3), si effettua un campionamento degli elementi rispetto ad una generica v.a. $\mathbb{X}_\sigma \sim \mathcal{D}$.

3.4.1 Generazioni con Apache Math

Apache Commons Math è una libreria di componenti matematici e statistici leggeri e indipendenti che affrontano i problemi più comuni non disponibili nel linguaggio di programmazione Java. L'uso che se n'è fatto in questa tesi è limitato al modulo dedicato alle distribuzioni di probabilità discrete, specifiche per variabili aleatorie in spettro a valori interi. La documentazione è piuttosto scarsa, tuttavia i sorgenti sono liberi e facilmente reperibili.

Questa libreria consente di definire il proprio seed di generazione pseudo-casuale. Requisito che può tornare utile, qualora risultasse necessario replicare un esperimento. Questo package risulta molto intuitivo. Basta infatti allocare un'oggetto passandogli 3 valori, quali

- seed
- probabilità empiriche
- spettro della variabile aleatoria

dopodiché l'oggetto allocato permetterà di generare degli elementi del dominio della v.a. coerentemente con la probabilità fornita.

3.5 Programmazione Dinamica per LCS

Tutte le meta-euristiche presentate calcolano almeno una volta una soluzione ammissibile per il problema `RFLCS` passando per un rilassamento per eliminazione. Eliminando i vincoli di *repetition-free* si ottiene il problema `LCS`. Naturalmente in generale, risolvere il problema rilassato, non porta ad una soluzione ammissibile per il problema originale. Offre però la possibilità di calcolare una soluzione in tempo polinomiale sulla taglia dell'istanza. Se si applica il rilassamento per risolvere sotto-istanze particolari si può tentare di ottenere delle soluzioni quantomeno "vicine" all'essere ammissibili, ossia con poche ripetizioni.

Questa sezione è dedicata alla stesura dell'algoritmo di programmazione dinamica utilizzato per il calcolo di una Longest Common Subsequence.

Algorithm 9 Compute optimal values matrix

```
1: procedure CreateMatrix(x, y)
2:     for i ← 0 to n do
3:         for j ← 0 to m do
4:             if i = 0 ∨ j = 0 then M[i, j] ← 0
5:             else if x_i = y_j then M[i, j] ← M[i − 1, j − 1] + 1
6:             else M[i, j] ← max{M[i − 1, j], M[i, j − 1]}
7:     return M
```

Il primo passo consiste nel calcolare una matrice di valori ottimi per tutte le sotto-istanze definite secondo una criterio di restrizione che goda di una sotto-struttura ottima. Ai fini della dimostrazione, si indica con $\mathcal{I}(n, m)$ un'istanza al problema `LCS`, formata da due sequenze $x_1x_2 \cdots x_n$ e $y_1y_2 \cdots y_n$ con $n, m \in \mathbb{N}$. Il criterio segue dal prossimo teorema:

Teorema. *$\mathcal{I}(n, m)$ gode di una sotto-struttura ottima.*

Dimostrazione. Sia $s = (s_1s_2 \cdots s_k)$ una soluzione ottima per $\mathcal{I}(n, m)$. Si verificano le seguenti affermazioni:

1. $x_n = y_m = \sigma \rightarrow z_k = \sigma$ e $s(1 : k - 1)$ è ottima per $\mathcal{I}(n - 1, m - 1)$
2. $x_n \neq y_m \wedge z_k \neq x_n \rightarrow s$ è ottima per $\mathcal{I}(n - 1, m)$
3. $x_n \neq y_m \wedge z_k \neq y_n \rightarrow s$ è ottima per $\mathcal{I}(n, m - 1)$

Il caso (1.) si dimostra in due passi: (i) si supponga per assurdo che $z_k \neq \sigma$, ma allora è possibile aggiungere σ stesso a destra di s, ottenendo una nuova soluzione ammissibile di grandezza $k + 1$, ossia migliore dell'ottimo, contraddizione. Quindi necessariamente $z_k = \sigma$. Resta da dimostrare che $s(1 : k - 1)$ è soluzione ottima per $\mathcal{I}(n-1, m-1)$. (ii) Anche in questo caso si procede per assurdo, supponendo esista una soluzione per l'istanza ridotta, detta $\bar{s}$, tale che $|\bar{s}| > k - 1$. Ma allora $\bar{s}\sigma$ è soluzione ammissibile per $\mathcal{I}(n, m)$ e $|\bar{s}\sigma| > k$, contraddizione.

Passando al caso (2.) si nota subito che, siccome $z_k \neq x_n$, s, oltre ad essere soluzione per $\mathcal{I}(n, m)$, è soluzione anche per $\mathcal{I}(n - 1, m)$. Una soluzione $|\bar{s}| > |s|$ per la sotto-istanza contraddice direttamente l'ipotesi che s è ottima per $\mathcal{I}(n, m)$.

Infine, per dimostrare il caso (3.), si procede simmetricamente a quanto fatto per il caso precedente.

□

L'algoritmo 9 implementa quindi la seguente definizione ricorsiva del valore ottimo di funzione obiettivo per `LCS`. Siano x e y due sequenze e sia $\mathcal{L}(n, m)$ la lunghezza di una `LCS` delle due sotto-stringhe ottenute da x e y considerando rispettivamente i primi

n ed m elementi. Formalmente

$$\mathcal{L}(n,m) = \begin{cases} 0 & n = 0 \vee m = 0 \\ \mathcal{L}(n-1, m-1) + 1 & \text{se } x_n = y_m \\ \max\{\mathcal{L}(n, m-1), \mathcal{L}(n-1, m)\} & \text{altrimenti.} \end{cases} \tag{3.2}$$

A partire dalla matrice costruita con l'algoritmo 9 si può ricavare una stringa che sia soluzione ottima per l'istanza originale, il cui valore di funzione obiettivo è memorizzato nell'ultima cella della matrice. Il procedimento è descritto dall'algoritmo 10.

Algorithm 10 Longest Common Subsequence Reconstruction

```
1: procedure LCS(x, y, M)
2:     s ← ϵ
3:     i ← |x|, j ← |y|
4:     while i > 0 ∧ j > 0 do
5:         if x_i = y_j then s ← s · x_i
6:         else if M[i − 1, j] > M[i, j − 1] then i ← i − 1
7:         else j ← j − 1
8:     return s
```

Si noti che il calcolo della matrice $\mathcal{M}$ ha complessità computazionale dominante. Questa fase consta di due cicli innestati, con una complessità pari ad $\mathcal{O}(|x| \cdot |y|)$.

3.6 Strutture Dati

3.6.1 Alfabeto

Le lettere di un'alfabeto sono codificate tramite numeri interi. Un alfabeto di cardinalità $|\Sigma| = n$ consiste in un insieme

$$\Sigma = \{0, 1, \ldots, n-1\}$$

L'encoding dell'alfabeto può coincidere quindi con n, la sua cardinalità.

3.6.2 Istanza

Si sono implementate due diverse strutture dati per la rappresentazione di una istanza per il problema.

Coppia di Array

Due array di interi permettono una semplice codifica di un'istanza $\mathcal{I}$, così come la codifica di una generica sotto-istanza $\mathcal{I}' \subseteq \mathcal{I}$. Essendo l'alfabeto definito con i soli numeri interi positivi, è possibile codificare $\mathcal{I}'$ sovrascrivendo le occorrenze $\mathcal{I} \setminus \mathcal{I}'$ con un valore negativo (-1). Tale struttura torna utile dal momento in cui è necessario tenere traccia delle posizioni originali delle occorrenze, ad esempio durante un merge tra due sotto-istanze [3.2]. Per comprendere meglio si supponga di avere un'istanza composta da due stringhe di 5 caratteri, la coppia di array risultante potrebbe essere la seguente

1	0	3	1	2
0	3	3	2	2

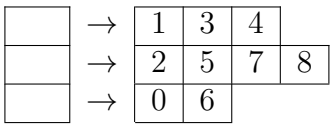
una sotto-istanza per questa coppia potrebbe essere

1	-1	-1	1	2
-1	3	3	-1	-1

Mappa delle Posizioni

Tramite una matrice irregolare (*jagged array*) è possibile codificare una mappa delle posizioni delle occorrenze di ogni singola lettera. Ricordando l'encoding dell'alfabeto, accedendo ad una posizione $a \in \Sigma$ nella prima dimensione della struttura, si ottiene un'array delle posizioni delle occorrenze di a in una generica stringa. Le posizioni hanno ordine crescente, così da permettere un calcolo più efficiente nel caso medio per la ricerca del minimo valore superiore ad una certa soglia.
Si supponga di avere un encoding di una sequenza di taglia 9, definita in un alfabeto di

3 caratteri, ad esempio 201001211. Il layout di memoria corrispondente sarà il seguente



Utilizzando due mappe così definite è possibile rappresentare un'istanza $\mathcal{I} = (x, y)$.

3.6.3 Lista dei Candidati

La lista $\mathcal{L}$ usata nel BR-GRASP è codificata tramite una rappresentazione logica dei candidati. Di fatto, la struttura è molto simile alla mappa descritta precendentemente. Una prima dimensione codifica l'accesso agli insiemi di occorrenze divisi per lettera, esattamente come per la mappa. In questo caso però in un'insieme, di una generica lettera dell'alfabeto, sono conservate le posizioni di entrambe le stringhe (x, y). L'insieme in questione sara quindi codificato tramite una coppia di array, di eguale taglia. In uno sono presenti le posizioni, mentre l'altro dichiara la stringa di appartenenza. In fase di allocazione, gli insiemi vengo ordinati secondo il criterio greedy.

3.6.4 Soluzione

Anche in questo caso sono state utilizzate due diverse strutture dati per la rappresentazione della soluzione.

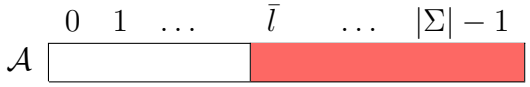
Coppia di Array

La prima è proprio la coppia di array, descritta precedentemente, utilizzata per la rappresentazione di una sotto-istanza. Utilizzando questa struttura si ha una codifica esplicita delle variabili di decisione che si assumono in soluzione, a partire da un'istanza espressa come insieme di coppie Z.

Singolo Array

Un array, detto $\mathcal{A}$, lungo esattamente $|\Sigma|$, accompagnato da un valore $\bar{l}$ che indica la lunghezza della soluzione memorizzata. Ciò significa che i primi $\bar{l}$ valori saranno

una sequenza di caratteri repetition-free e sottosequenza di entrambe le stringhe (x, y). Le lettere non presenti in soluzione, anziché non essere prese in considerazione, sono memorizzate nel resto dell'array, a partire dalla posizione $\bar{l}$.



Questa struttura facilita le fasi di ricerca locale e diversificazione. Inoltre, mentre con la struttura precedentemente descritta si teneva traccia di quali occorrenze si fossero scelte, in questo caso l'informazione si perde. L'idea è che tale informazione sia troppo restrittiva, nel momento in cui occorre effettuare una ricerca locale per inserimenti. Sacrificando quindi l'efficienza durante la check validity, si massimizza la probabilità che un'operazione di inserimento vada a buon fine.

Capitolo 4

Analisi

4.1 Descrizione Istanze

Per la sperimentazione si è utilizzato un data-set massiccio fornito dal Dr. Christian Blum ed utilizzato per la sperimentazione in [20]. Trattasi di 4000 istanze classificabili per tipologia e taglia. Un file di una generica istanza riporta:

- encoding dell'alfabeto
- 2 stringhe accompagnate dalla relativa lunghezza

Le tipologie di dati risultano essere un'estensione di quanto già fatto in [1]. Si hanno infatti due macro classi di istanze.

- generazione totalmente random
- numero di occorrenze limitato superiormente

Il primo insieme consiste di istanze suddivise per taglia e dimensione dell'alfabeto. In particolare le dimensioni scelte sono tutte potenze di 2. Si ha quindi che un'istanza $\mathcal{I} = (x, y)$ di quest'insieme è tale che $|x| = |y|$ e le dimensioni previste sono $\{32, 64, 128, 256, 512, 1024, 2048, 4096\}$. L'alfabeto è espresso come frazione della taglia dell'input, ovvero $\{\frac{n}{8}, \frac{n}{4}, \frac{3n}{8}, \frac{n}{2}, \frac{5n}{8}, \frac{3n}{4}, \frac{7n}{8}\}$, con n dimensione delle stringhe.
Il secondo insieme e formato da istanze generate in maniera random ad eccezione di una

limitazione nel numero di ripetizioni di ogni lettera. L'insieme in questo caso è classificato per taglia dell'alfabeto, $\{4, 8, 16, 32, 64, 128, 256, 512\}$, e per frequenza massima $f_{max} \in \{3, 4, 5, 6, 7, 8\}$.

4.2 Verifica Intrattabilità del Problema

Quando si affronta un problema intrattabile, è utile avere un riscontro di cosa significhi in un contesto reale. A partire da una formulazione matematica è possibile affrontare il relativo problema con approcci general purpose per problemi di programmazione matematica (nel caso del `RFLCS` trattasi di programmazione lineare intera). Si è quindi usato l'ottimizzatore incluso all'interno del framework IBM ILOG CPLEX Optimization Studio.

IBM ILOG CPLEX Optimization Studio

Il framework consente di definire un problema matematico tramite il linguaggio di programmazione OPL (Optimization Programming Language). Una volta codificato il problema, è possibile sottoporre un insieme di istanze all'ottimizzatore matematico CPLEX, che proverà a risolverle tramite approccio branch-and-cut.

Branch and Cut

Nasce come unione di due tecniche:

- Branch and Bound
- Piano di Taglio

L'idea alla base è di ottenere o una soluzione intera o un nuovo lower bound. Se ciò non avviene si esegue *branching*.

La definizione di efficaci procedure di separazione è il punto cruciale dell'algoritmo; per esse esistono due diverse metodologie di sviluppo:

- procedure di separazione general-purpose, applicabili a ogni generico problema di PLI. É il caso di CPLEX, qui utilizzato per risolvere `RFLCS`.

- procedure di separazione specifiche per una data classe di problemi, si sviluppa cioè una procedura ad hoc per il problema. Come ad esempio accade in [18].

In generale la complessità dell'algoritmo è super-polinomiale, siccome il numero di branching è esponenziale rispetto alla taglia delle istanze. Una stima precisa dell'algoritmo ha senso dal momento in cui si definisce un metodo di risoluzione di un sotto-problema ed un metodo di risoluzione per il problema di separazione. Ad ogni modo, anche sotto l'ipotesi che le componenti richiedano tempo costante, l'algoritmo branch-and-cut ha senso di essere utilizzato esclusivamente per problemi intrattabili.

Risulati

$\frac{\lvert\Sigma\rvert}{n}$	n/8	n/4	3n/8	n/2	5n/8	3n/4	7n/8
32	4	7,83	8,77	8,87	8,6	8,17	7,67
64	8	14,67	15,53	14,8	13,3	12,53	11,57
128	15,13	25,93	24,87	21,93	21,1	19,7	18,4
256	n.a.	38,63	39,9	35,2	32,53	29,93	26,53
512	n.a.	n.a.	5,5	10,7	25,23	21,97	33,13
1024	n.a.	n.a.	n.a.	n.a.	n.a.	0,03	1,17
2048	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
4096	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.

Tabella 4.1: Risoluzione istanze del primo tipo (sequenze casuali). Dove si trova la dicitura n.a. (not available) vuol dire che l'esecuzione non è andata a buon fine a causa di un'allocazione di memoria superiore alle possibilità del computer.

Risulta chiaro dalla tabella 4.1 che il problema in esame diventa rapidamente insostenibile per il computer, che già con sequenze di 256 caratteri e con l'alfabeto più piccolo (che implica un elevato numero di variabili di decisioni) non riesce a completare il calcolo.

Esito simile si è ottenuto durante la batteria di esecuzioni sul data-set formato da istanze con ripetizioni limitate superiormente - tabella 4.2. Si osservi che, più ripetizioni sono ammesse, maggiori saranno le variabili di decisione che l'ottimizzatore dovrà allocare. Risulta chiaro perchè la macchina mostra i primi segni di cedimento sulle istanze con massimo 7 ripetizioni e taglia dell'alfabeto pari a 64. Si ricordi che il valore atteso

$\lvert\Sigma\rvert$ / f_{max}	4	8	16	32	64	128	256	512
3	3,47	6,23	9,70	16,13	25,43	36,77	n.a.	n.a.
4	3,77	6,87	11,57	19	30,37	44,53	n.a.	n.a.
5	3,83	7,40	12,93	21,6	34,93	20,33	n.a.	n.a.
6	3,90	7,53	14,00	23,73	34,03	n.a.	n.a.	n.a.
7	3,97	7,70	14,93	25,57	n.a.	n.a.	n.a.	n.a.
8	3,97	7,77	14,80	27,5	n.a.	n.a.	n.a.	n.a.

Tabella 4.2: Risoluzione istanze del primo tipo (sequenze casuali con ripetizioni limitate).

per la taglia delle sequenze di questa tipologia è pari a 224, valore simile alla lunghezza delle istanze del primo tipo dove non si è ottenuta una soluzione.

4.3 Generazione Training Set

Il training set è l'insieme d'istanze destinato ad essere utilizzato in fase di tuning, per la valutazione dei parametri. É buona norma utilizzare delle istanze diverse da quelle riservate alla fase di sperimentazione finale. Ciò permette di evitare il cosidetto *overfitting*, ossia un sovradattamento dei parametri rispetto ai campioni in possesso, con conseguente perdita di generalità dell'algoritmo.

Si sono implementati due diversi metodi di campionamento, corrispondenti alle due macro-tipologie di istanze. Un primo metodo prende in input la lunghezza delle stringhe e l'encoding dell'alfabeto, restituendo in output delle coppie di stringhe casuali della lunghezza desiderata e con dominio nell'alfabeto dato. Il secondo metodo, leggermente più complesso, genera stringhe del secondo tipo. L'algoritmo sceglie, per ogni lettera dell'alfabeto, il numero di ripetizioni all'interno della sequenza, rispettando il valore massimo dato. É possibile effettuare il campionamento in tempo lineare sul valore atteso della taglia [1], ovvero

$$\mathbb{E}[l_x] = \mathbb{E}[l_y] = \frac{|\Sigma|}{2} \cdot f_{max}$$

con f_{max} frequenza massima. I metodi sopra descritti sono stati infine utilizzati per la creazione del training set, composto da due sotto-insiemi.

Il primo insieme è composto da istanze formate da sequenze casuali. In particole tutte le sequenze hanno lunghezza 4096. Scegliendo una taglia grande si vuole generare un campione che rappresenti bene le istanze casuali. Si considerano quindi della stessa tipologia istanze di dimensione diversa, ma uguale rapporto $\frac{|\Sigma|}{n}$ con dimensione $n \in \{32, 64, 128, 256, 512, 1024, 2048, 4096\}$. Quest'insieme è quindi formato da istanze con dimensione dell'alfabeto $|\Sigma| \in \{\frac{n}{8}, \frac{n}{4}, \frac{3n}{8}, \frac{n}{2}, \frac{5n}{8}, \frac{3n}{4}, \frac{7n}{8}\}$ con $n = 4096$. In particolare si sono generate 2 istanze per ogni dimensione dell'alfabeto.
Il secondo insieme del training set è composto dalle istanze formate da sequenze causali con ripetizioni limitate superiormente. Similmente al caso precedente, qui il training set è composto da sole istanze con $|\Sigma| = 512$. Le tipologie qui si differenziano per frequenza massima f_{max}. Si osservi che, fissato f_{max}, si verifica per qualsiasi $|\Sigma|$ che

$$\frac{|\Sigma|}{n} = \frac{2|\Sigma|}{|\Sigma| \cdot f_{max}} = \frac{2}{f_{max}}$$

Vale a dire che il rapporto tra dimensione dell'alfabeto e valore atteso della lunghezza delle sequenze dell'istanza è costante. Per questo motivo, come prima si considerano della stessa tipologia le istanze con stessa f_{max}. L'insieme sarà quindi composto da 2 istanze per ogni $f_{max} \in \{3, 4, 5, 6, 7, 8\}$.

4.4 Tuning con irace

Il pacchetto irace implementa una procedura di *iterated procedure*, un'estensione di Iterated F-race [27]. Il pacchetto si usa principalmente per la configurazione automatica dei parametri per algoritmi di ottimizzazione, cioè, trovare le impostazioni più appropriate di un algoritmo dato un insieme di istanze di un problema [28]. É implementato a partire dal pacchetto race ed è scritto in **R**. Irace chiede in input 3 diversi parametri quali

- istanze di training
- spazio dei parametri
- meta-parametri

Le istanze di training sono quelle su cui il motore configurerà i parametri. Per questi ultimi è necesserio definire un dominio o, appunto, spazio dei parametri. Il pacchetto permette di dichiarare parametri definiti in intervalli di valori continui, discreti oppure enumerazioni. Infine occorre fornire dei valori per i meta-parametri, ovvero una configurazione per lo scenario di lavoro del framework. Ad esempio è possibile scegliere il budget, in termini di secondi concessi al tuning o in termini di numero di esecuzioni. O ancora si può dichiarare la precisione macchina per i parametri reali (numero di cifre decimali significative).
Una volta definito l'ambiente di esecuzione, da R è possibile eseguire il **targetRunner**. Il processo effettuerà numerose chiamate all'eseguibile dell'algoritmo che si vuole configurare, passandogli in input

- un seed di default random
- parametri correnti

Al termine, è possibile visualizzare diversi vettori configurazione, considerati soluzioni elite.

4.5 Valutazione Costruzione Greedy

Durante la progettazione di un'algoritmo può essere utile valutare le sue componenti come moduli indipendenti. Si è quindi effettuata una valutazione preliminare del processo di costruzione di una soluzione utilizzato negli algoritmi GRASP ed ILS. Ai fini della valutazione si fa riferimento all'algoritmo evolutivo (dicitura EA), presentato da Castelli et al. in [19] nel 2013. L'algoritmo è già stato oggetto di comparazione con l'algoritmo attuale stato dell'arte, i risultati sperimentali sono reperibili nell'articolo di C. Blum e Maria J. Blesa pubblicato nel 2017 [20].

Focalizzandosi sulle istanze di taglia più grande si può notare come l'algoritmo riesca persino a superare i rusltati dell'EA. Tale comportamento trova spiegazione se si pensa all'intuizione di natura probabilistica su cui è basato il criterio greedy. Nell'ottica di valutare la bontà della funzione greedy ci si focalizza su tipologie di istanze dove il

criterio perde o guadagna maggior robustezza. Le tipologie in questione sono le istanze con cardinalità dell'alfabeto pari ad 1/8 della taglia, come caso peggiore, e 7/8 come caso migliore. Il primo caso, oltre che essere più complesso a causa di uno spazio di ricerca potenzialmente molto esteso, si pensa causi anche un appiattimento dei valori greedy, nel senso che gli elementi candidati ad entrare in soluzione hanno lo stesso grado di appetibilità rendendo complessa la scelta. Ciò si traduce in distribuzioni quasi equiprobabili. Nel secondo caso si parla di un alfabeto molto grande, Se rapportato all'istanza. Per questo motivo è verosimile che un campione casuale presenti molte lettere dell'alfabeto per le quali non esiste una soluzione di cui possano far parte, in termini dell'insieme di coppie utilizzato per la formulazione matematica, ci si aspetta che un insieme considerevole di lettere, detto $\Sigma' \subset \Sigma$, sia tale che $\forall \sigma \in \Sigma', Z_\sigma = \emptyset$. Ma più in generale si pensa che per un numero importante di lettere dell'alfabeto, il relativo sotto-insieme di coppie abbia una cardinalità contenuta, con un conseguente spazio di ricerca relativamente semplice da esplorare. Il calcolo delle probabilità associate ai candidati dovrebbe giovare di queste proprietà, manifestando delle distribuzioni più eterogenee rispetto alle altre tipologie definite. Sperimentalmente ci si è accertati, con esito positivo, che queste considerazioni fossero fondate.

Ricapitolando, nei grafici riportati nelle figure 4.2 e 4.1 si confrontano i valori di funzione obiettivo ottenuti da EA e componente di costruzione - dicitura GREEDY - sulle tipologie scelte e precedentemente motivate. Da entrambi i grafici si evince che GREEDY riesce a competere con EA. Si dimostra inoltre che GREEDY è più robusto di EA al aumentare della taglia, riuscendo a vincere su entrambe le tipologie con taglia massima. In definitiva il metodo di costruzione, ed in particolar il criterio greedy su cui è basato, si è rivelata una componente dal potenziale notevole, senz'altro addatta ad essere inserita in framework più estesi.

4.6 Valutazione Metaeuristiche

In quest'analisi ci si pone l'obiettivo di decretare un algoritmo vincente tra quelli implementati. Si procederà dunque per due step. Inizialmente si effettua una prima

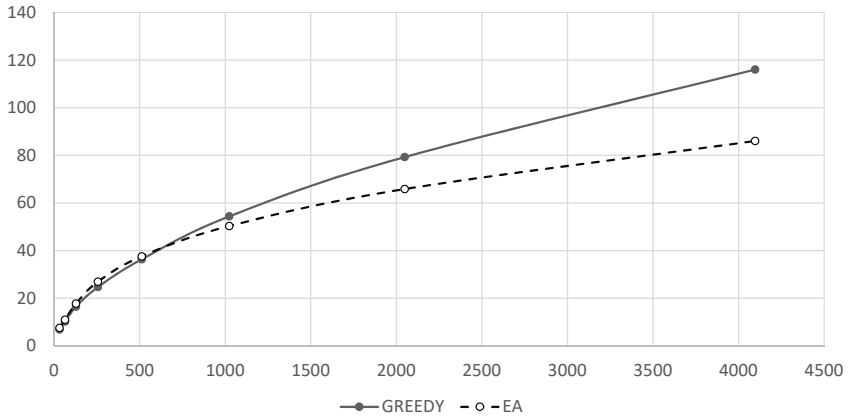


Figura 4.1: valori medi di funzione obiettivo ottenuti da GREEDY ed EA sulle istanze della tipologia $|\Sigma| = \frac{7n}{8}$ con $n \in \{32, 64, 128, 256, 512, 1024, 2048, 4096\}$. L'asse verticale è il dominio dei valori medi, mentre l'asse orizzontale indica la taglia delle istanze.

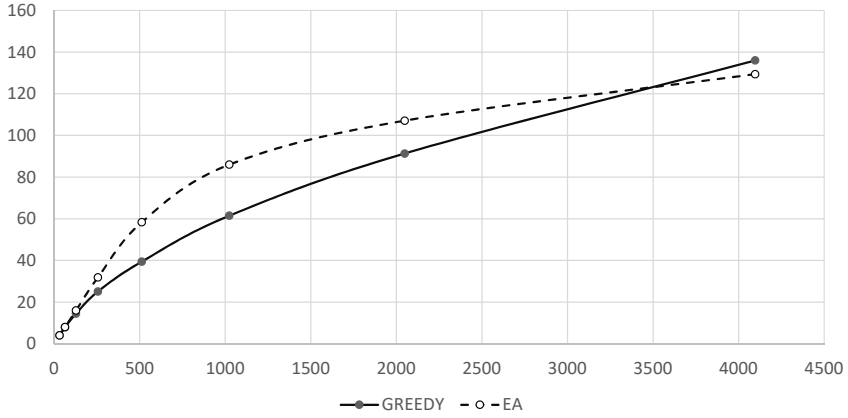


Figura 4.2: valori medi di funzione obiettivo ottenuti da GREEDY ed EA sulle istanze della tipologia $|\Sigma| = \frac{n}{8}$ con $n \in \{32, 64, 128, 256, 512, 1024, 2048, 4096\}$.

comparazione tra GRASP e ILS. Successivamente, l'algoritmo migliore concorrerà con il Generate and Solve. Gli algoritmi si valuteranno in base agli esiti ottenuti sullo stesso data-set, i cui risultati sono riportati nella sezione A. Per ogni tipologia e taglia d'istanza si confrontano i due valori media e se ne misura il *gap*. Formalmente siano $\mathcal{P}$ e $\mathcal{Q}$ due algoritmi, si vuole calcolare il gap tra due valori di funzione obiettivo $\mathcal{P}(x, y)$ e $\mathcal{Q}$, per un'istanza (x, y). Supponendo di voler sapere di quanto $\mathcal{P}$ migliora rispetto a $\mathcal{Q}$, la funzione gap sarà

$$\text{gap}(\mathcal{P}, \mathcal{Q}, x, y) = \begin{cases} 1 - \frac{\mathcal{Q}(x,y)}{\mathcal{P}(x,y)} & \text{se } \mathcal{P}(x, y) > \mathcal{Q}(x, y) \\ \frac{\mathcal{P}(x,y)}{\mathcal{Q}(x,y)} - 1 & \text{altrimenti.} \end{cases}$$

Si è definità una funzione $gap : \mathbb{R}^+ \times \mathbb{R}^+ \to [-1, +1]$ i cui valori negativi indicano un peggioramento di $\mathcal{P}$ rispetto ad $\mathcal{Q}$, mentre i positivi corrispondono ad un miglioramento.

Comparazione GRASP-ILS

In prima analisi si è scelto di comparare GRASP ed ILS, in quanto algoritmi con alcuni punti in comune, quali il metodo di costruzione ed il metodo di ricerca locale. Come punti di distacco troviamo invece la modalità d'uso della componente di costruzione, che nel GRASP viene sfruttata ad ogni iterazione, mentre in ILS viene usata solo una volta, come punto di partenza per l'algoritmo. Altro punto di distacco sarà quindi l'approccio di diversificazione. Il GRASP in quanto algoritmo multi-start effettua numerose costruzioni indipendenti con un certo grado di diversità, fortemente relato alla distribuzione di probabilità dei candidati. Invece l'ILS applica delle eliminazioni allo scopo di allontanarsi da ottimi locali.

Ci si chiede dunque se GRASP sia migliore di ILS. I risultati sono riportati nella figure 4.3 e 4.4. Risulta piuttosto chiaro quale sia l'algoritmo migliore. Il GRASP ottiene un miglioramento quasi ovunque con picchi del 9%. Il gap raggiunge i valori massimi su istanze di taglia 1024, mentre comincia a ridursi per sequenze più lunghe. Un comportamento anomalo avviene per la tipologia $|\Sigma| = \frac{n}{8}$. In questo caso, che è il più complesso da risolvere, l'ILS risulta migliore di circa il 2% su istanze di taglia 512 e 1024. Tuttavia al crescere delle istanze GRASP si conferma migliore anche in questo

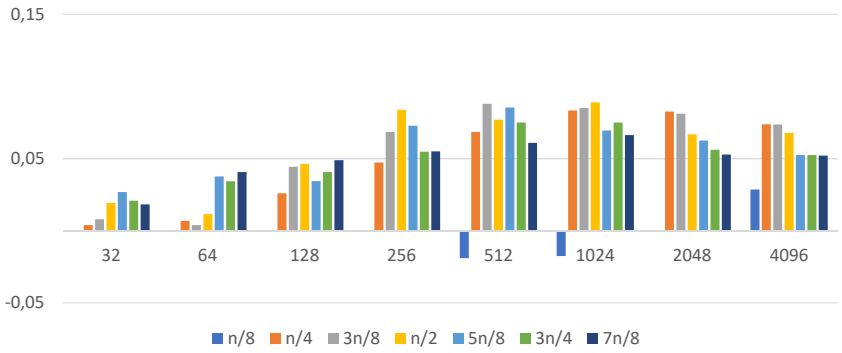


Figura 4.3: confronto GRASP ed ILS su istanze del primo data set (sequenze random). L'asse verticale esprime il miglioramento di GRASP rispetto ad ILS. Sull'asse orizzontale si raggruppano i valori per taglia delle istanze. Infine la legenda indica la tipologia di istanza, in termini di $|\Sigma| = n$.

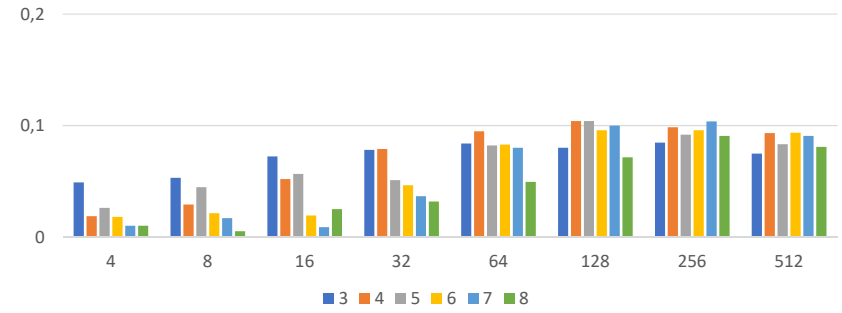


Figura 4.4: confronto GRASP ed ILS su istanze sul secondo data set (sequenze random con frequenza limitata superiormente). L'asse verticale esprime il miglioramento di GRASP rispetto ad ILS. Sull'asse orizzontale si raggruppano i valori per cardinalità dell'alfabeto. Infine la legenda indica la tipologia di istanza, in termini di f_{max}.

caso, rendendo evidente quale sia il vincitore. Il secondo data set consolida quanto detto. In questo caso (figura 4.4) il GRASP vince in tutti i casi.

Comparazione GS-GRASP

Dalla comparazione precedente si è osservato che l'algoritmo GRASP è migliore dell'ILS per quasi tutte le istanze. La prossima comparazione riguarda quindi GRASP e GS. In particolare ci si chiede se GS sia migliore di GRASP. Dai grafici in figura 4.5 e 4.6 non emerge distinamente un algoritmo più efficiente. GRASP converge a valori di funzione obiettivo superiori su istanze di taglia piccola e media. Il gap si inverte a partire dalle sequenze di 1024 caratteri. Il CMSA è dunque più robusto al crescere dell'istanza così come al decrescere dell'alfabeto, raggiungendo un gap del 15% sulle istanze $l_x = l_y = 4096$ e $|\Sigma| = 512$.

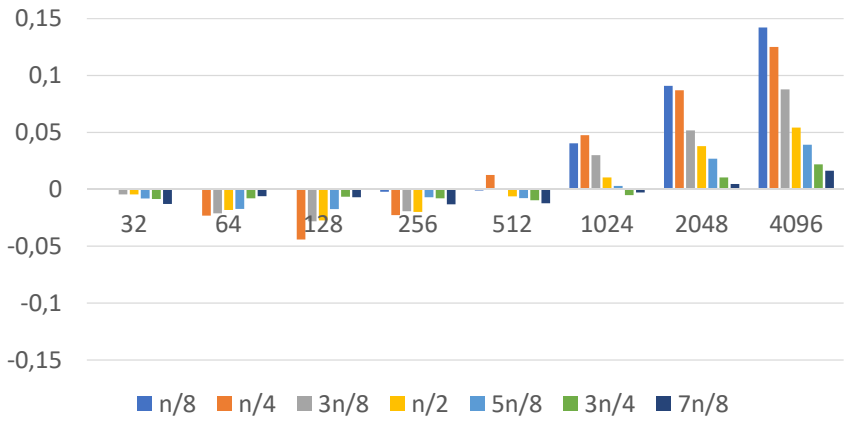


Figura 4.5: gap di miglioramentro di GS rispetto a GRASP sul primo data set.

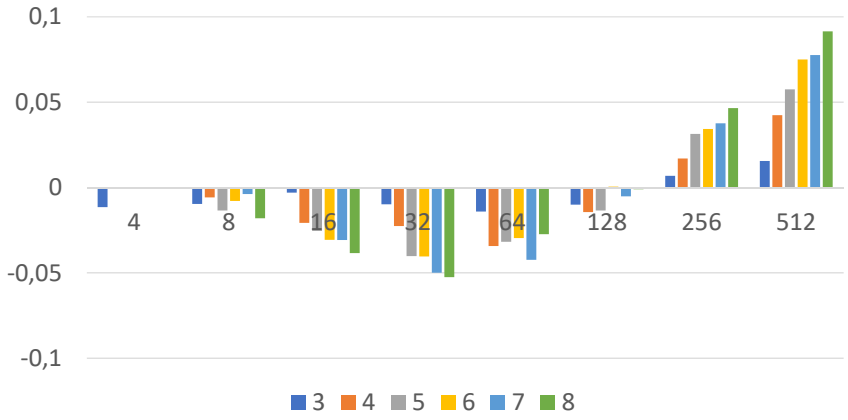


Figura 4.6: gap di miglioramentro di GS rispetto a GRASP sul secondo data set.

4.7 Comparazione con Hyb-CMSA

In questa sezione si propone una comparazione dei migliori algoritmi, implementati durante l'attività di tesi, con lo stato dell'arte. Trattasi dell'algoritmo Hyb-CMSA presentato nel 2017 (sezione 2.3.4). Con questa analisi ci si pone l'obiettivo di aprire strade che abbiano il potenziale per rinnovare lo stato dell'arte attuale. Si farà quindi riferimento ai valori di funzione obiettivo ottenuti da Hyb-CMSA sullo stesso data-set fino ad ora utilizzato.

Prima di procedere con la comparazione tra tabelle, va fatta una premessa riguardo le differenze tecnologiche. Gli algoritmi proposti in questa tesi sono stati eseguiti da una macchina con processore Intel Core i5 e 4GB di memoria RAM. Mentre Hyb-CMSA è stato testato su un sistema decisamente più performante, ossia un cluster di computer con Intel Xeon x5670 con non meno di 40GB di RAM.
Inoltre il budget di tempo necessario ad ILS, GRASP e GS è nettamente inferiore ai tempi richiesti da Hyb-CMSA. Il divario costante, osservabile nelle tabelle 4.4 e 4.3, è quindi consolidato dalla differenza prestazionale delle macchine utilizzate. Questo confronto ha il solo scopo di mettere alla luce il potenziale dei nuovi algoritmi, dal punto

di vista del running-time. Ad esempio si può notare come, al crescere della taglia, il budget cresce in maniera molto contenuta. Vale lo stesso se si guarda orizzontalmente, al crescere di $n - |\Sigma|$.

GRASP	n/8	n/4	3n/8	n/2	5n/8	3n/4	7n/8
512	1,92	1,79	1,66	1,53	1,42	1,28	1,15
1024	3,84	3,58	3,33	3,07	2,82	2,56	2,30
2048	7,68	7,17	6,66	6,15	5,63	5,12	4,61
4096	15,37	14,36	13,33	12,32	11,28	10,27	9,24

HYB-CMSA	n/8	n/4	3n/8	n/2	5n/8	3n/4	7n/8
512	16	10	7	7	3	3	3
1024	73	32	20	22	11	4	14
2048	160	84	58	25	44	30	43
4096	272	234	385	106	200	71	107

Tabella 4.3: i valori fanno riferimento al numero di secondi impiegato da GRASP e HYB-CMSA per la risoluzione di istanze di taglia media e grande. Il distacco raggiunge i 2 ordini di grandezza, con il caso 4096 - 7n/8.

GRASP	3	4	5	6	7	8
128	1,76	1,02	1,27	1,54	1,80	2,02
256	1,53	2,04	2,56	3,09	3,55	4,08
512	3,05	4,09	5,13	6,15	7,13	8,21

HYB-CMSA	3	4	5	6	7	8
128	2	3	9	15	27	37
256	3	9	20	31	28	46
512	17	27	63	63	112	124

Tabella 4.4: tempi di calcolo impiegati da GRASP e HYB-CMSA per la risoluzione del secondo data-set ristretto alle istanze con $|\Sigma| \in \{128, 256, 512\}$.

Infine si osservi che tutti gli algoritmi in analisi, compreso HYB-CMSA, sono improntati al raggiungimento della soluzione migliore che sono in grado di trovare, in quanto il budget di tempo è, in tutti i casi, scelto per soddisfare tale convergenza. In questo modo i valori di funzione obiettivo, di cui si dispone, sono indipendenti dalla macchina. Per questo motivo si ritiene che, dedicarsi alla riproduzione degli esperimenti, porterebbe agli stessi valori teorici e non darebbe un significativo valore aggiunto alla comparazione.

Se si ottennessero dei risultati in qualche modo competitivi, in termini di bontà delle soluzioni, gli algoritmi proposti costituirebbero una valida alternativa allo stato dell'arte. Alternativa che potrebbe rivelarsi oltremodo necessaria in contesti applicativi dove la rapidità di risposta è una prerogativa. Si pensi, ad esempio, di dover processare una mole di dati ben superiore a quella del campione qui utilizzato. In ultima analisi si vuole quindi effettuare una comparazione con lo stato dell'arte. Nella sezione 4.6 si è visto che, durante la comparazione tra GRASP e GS, non è emerso un algoritmo più vantaggioso in assoluto. Si è piuttosto osservato che GRASP funziona meglio su istanze di taglia piccola e media, mentre GS è superiore durante il calcolo di istanze di grandi dimensioni. Per questo motivo si procede dividendo in due l'analisi. Prima si calcola il gap tra Hyb-CMSA e GRASP per istanze di taglia fino a 512, nel caso del primo insieme, e alfabeto di cardinalità fino a 128, nel caso del secondo insieme. Successivamente si calcola il gap tra Hyb-CMSA e GS per le istanze del primo insieme con taglia da 1024, 2048 e 4096, e per il secondo insieme con $|\Sigma| \in \{256, 512\}$.

Comparazione GRASP-HybCMSA

Dalle figure 4.7 e 4.8 si nota che l'algoritmo GRASP non solo si mantiene con dei gap bassi, ma in svariati casi eguaglia o oltrepassa la controparte. Nel caso del primo insieme, il GRASP risulta migliore di Hyb-CMSA, che riesce ad invertire gli esiti da taglia 256, mantenendo dei gap relativamente bassi, ad eccezione del caso $n = 512, |\Sigma| = \frac{n}{4}$. Similmente, sul secondo insieme GRASP si assesta come miglior algoritmo, mente Hyb-CMSA acquisisce distacco a partire da $|\Sigma| = 64$. In conclusione GRASP si rivela una valida alternativa a Hyb-CMSA con un gap medio inferiore al -1%, per le istanze considerate.

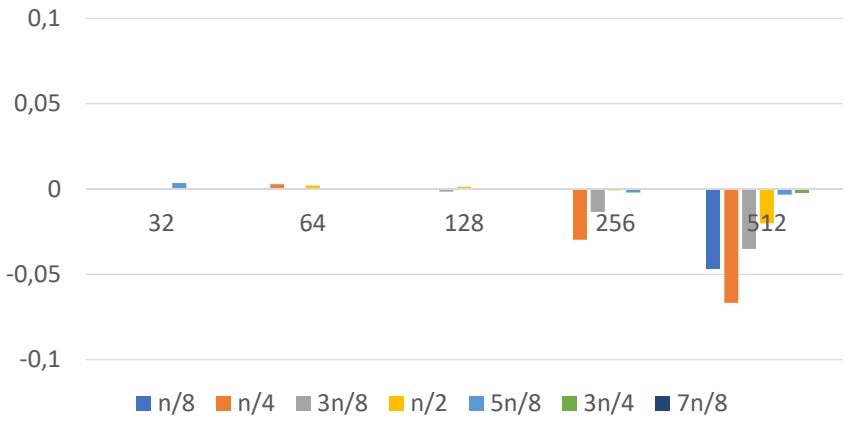


Figura 4.7: gap di miglioramento di GRASP rispetto a Hyb-CMSA sul primo data set, ridotto alle istanze di taglia piccola e media.

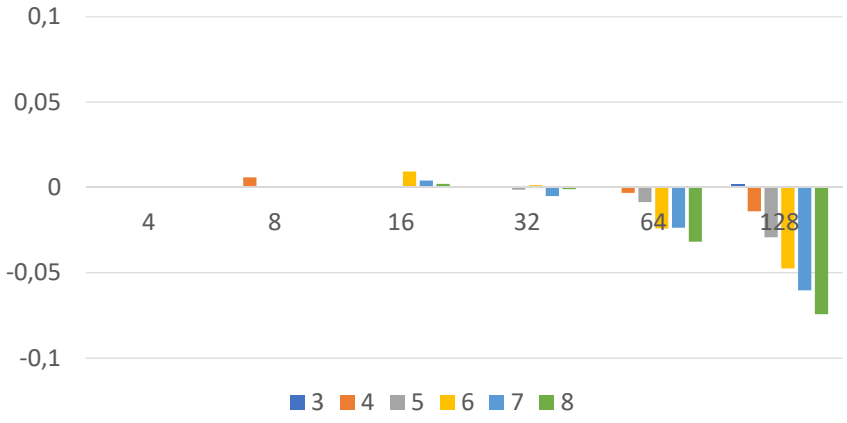


Figura 4.8: gap di miglioramento di GRASP rispetto a Hyb-CMSA sul secondo data set, ridotto alle istanze con cardinalità $|\Sigma| \in \{4, 8, 16, 32, 64, 128\}$.

Comparazione GS-HybCMSA

Si prosegue con il calcolo del gap tra GS ed Hyb-CMSA. Anche in questo caso si sono misurati dei valori piuttosto bassi, che in alcuni casi rasentano lo 0, ad esempio, per istanze della tipologia $n = 4096, |\Sigma| = \frac{7n}{8}$, GS ha ottenuto un valore di funzione obiettivo medio pari a 126,9 contro il 127,2 di Hyb-CMSA. In questo caso il gap medio è inferiore al 3%. Un sacrificio che può risultare accettabile se rapportato al guadagno in termini di tempi di calcolo. I gap ottenuti sono riportati nei grafici delle figure 4.9 e 4.10. In entrambi i casi si nota come il divario va a ridursi in maniera quasi monotona al crescere di $|\Sigma|$, inoltre solo sulla tipologia $n/8$ il gap supera il 4%, in riferimento al primo data set. Un altro fenomeno osservabile è che il gap medio, ottenuto sulle istanze di una taglia fissata, diminuisce costantemente al crescere dell'istanza, ad esempio per il primo data set il gap medio per istanze di taglia 1024, 2048 e 4096 è -0.026, -0.023 e -0.013, rispettivamente.

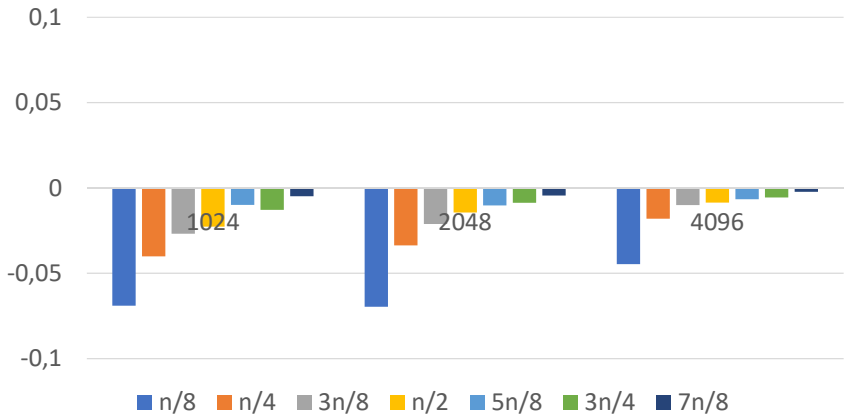


Figura 4.9: gap di miglioramento di GS rispetto a Hyb-CMSA sul primo data set, ridotto alle istanze di taglia grande.

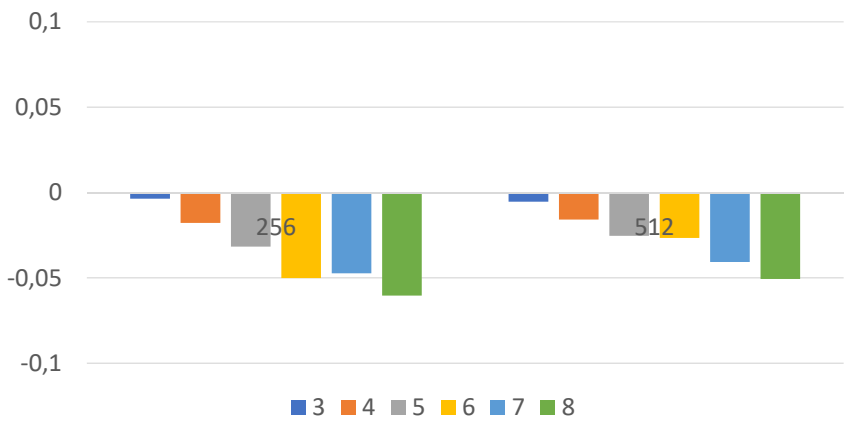


Figura 4.10: gap di miglioramento di GS rispetto a Hyb-CMSA sul secondo data set, ridotto alle istanze con cardinalità $|\Sigma| \in \{256, 512\}$.

4.8 Comparazione Algoritmi di Costruzione

Nella sezione precedente si è effettuata una comparazione tra 3 metaeuristiche, due delle quali appartengono al set di algoritmi progettati durante questo lavoro di tesi (Greedy Randomized Adaptive Search Procedure, Generate and Solve), mentre uno costituisce l'attuale stato dell'arte per via della bontà delle soluzioni che è in grado di ottenere (Hyb-CMSA).

Il confronto ha confermato la robustezza dell'algoritmo Hyb-CMSA, che riesce a ottenere valori di funzione obiettivo migliori della controparte su istanze grandi e medie. Tuttavia è interessante osservare che il gap è mai superiore all'8% e mediamente inferiore al 3%. Un risultato in parte soddisfacente, che offre ulteriori spunti di analisi. Si pensa infatti che GRASP e GS siano algoritmi competitivi grazie alla loro componente di costruzione, particolarmente efficace. Questa componente potrebbe essere migliore di quella sfruttata invece da Hyb-CMSA. Per questo motivo si propone una comparazione delle due componenti, o più precisamente della loro versione deterministica, dove quindi le scelte vengono effettuate in maniera puramente greedy, senza randomizzare alcuna scelta.

Per poter effettuare il confronto è stato necessario reimplementare l'algoritmo di costruzione, il cui procedimento è descritto dalla pseudocodifica 11.

Algorithm 11 BLUM'S DETERMINISTIC SOLUTION BUILDER

1: **procedure** BUILD(x, y)
2: $\quad s \leftarrow \emptyset$
3: $\quad C \leftarrow \{z_{i,j} \in Z \mid i \leq h \wedge j \leq k, \forall z_{h,k} \in Z_{x_i}\}$
4: $\quad$ **while** $C \neq \emptyset$ **do**
5: $\quad\quad z_{p,q} \leftarrow \underset{z_{h,k} \in C}{\operatorname{argmin}} \frac{h}{|x|} + \frac{k}{|y|}$
6: $\quad\quad s \leftarrow s \cup \{z_{p,q}\}$
7: $\quad\quad C \leftarrow \{z_{i,j} \in Z \mid p < i \leq h \wedge q < j \leq k, \forall z_{h,k} \in Z_{x_i}\}$
8: $\quad$ **return** s

Risultati

In prima analisi, ci si focalizza sui tempi di esecuzione impiegati da GREEDY e BLUM (tabelle 4.5, 4.6). Per una questione di leggibilità, si riportano esclusivamente i tempi corrispondenti all'esecuzione delle tipologie di istanza più lente e più veloci.

n	32	64	128	256	512	1024	2048	4096
GREEDY	.00053	.00103	.00053	.00087	.00123	.00683	.01647	.06723
BLUM	.00013	.00007	.00007	.00020	.00033	.00077	.00097	.00327

Tabella 4.5: running-time su istanze della tipologia $|\Sigma| = \frac{n}{8}$

n	32	64	128	256	512	1024	2048	4096
GREEDY	.00000	.00000	.00000	.00053	.00103	.00417	.01423	.05427
BLUM	.00000	.00003	.00003	.00017	.00040	.00133	.00410	.01263

Tabella 4.6: running-time su istanze della tipologia $|\Sigma| = \frac{7n}{8}$

Analisi

Facendo un'analisi dei costi in termini di running-time, si considerano i due metodi comparabili, soprattutto nel caso in cui l'algoritmo è destinato a far parte di una procedura più ampia e di conseguenza complessa (si pensi alla metaeuristica HYB-CMSA, il cui budget massimo è di diversi minuti). Entrambi gli algoritmi sono infatti

molto efficienti, e non richiedono mai più di 1 decismo di secondo per calcolare una soluzione.

Passando ai risultati in termini di valori di funzione obiettivo, BLUM è particolarmente efficace durante la risoluzione di istanze della tipologia $|\Sigma| = \frac{n}{8}$. Andando infatti a confrontare i valori con quelli ottenuti da GREEDY - l'algoritmo di costruzione utilizzato da GRASP e GS - si osservano dei valori migliori su tutte le taglie in considerazione. Si è già però notato come il nuovo criterio greedy subisca un'appiattimento al decrescere del rapporto $\frac{|\Sigma|}{n}$ e ciò spiega come mai la taglia con rapporto minimo porta a questo esito. Tuttavia già dalla tipologia $|\Sigma| = \frac{n}{4}$ l'esito è invertito e vede GREEDY come algoritmo migliore su ogni combinazione di tipologia e taglia. I grafici in figura 4.11 e 4.12 mostrano chiaramente, tramite calcolo del gap, come l'algoritmo GREEDY sia migliore dell'algoritmo BLUM, toccando picchi del 20% e con un gap di miglioramento medio del 10%.

Si conclude che sarebbe opportuno continuare ad investire sulla progettazione di un metodo che includa tale criterio (o sue relative varianti che sfruttino la funzione greedy, definita nella sezione 3.1.1). Ad esempio potrebbe portare a nuovi risultati imbattutti l'introduzione di GREEDY all'interno della metaeuristica Hyb-CMSA, sostituendo o alternando il criterio originale.

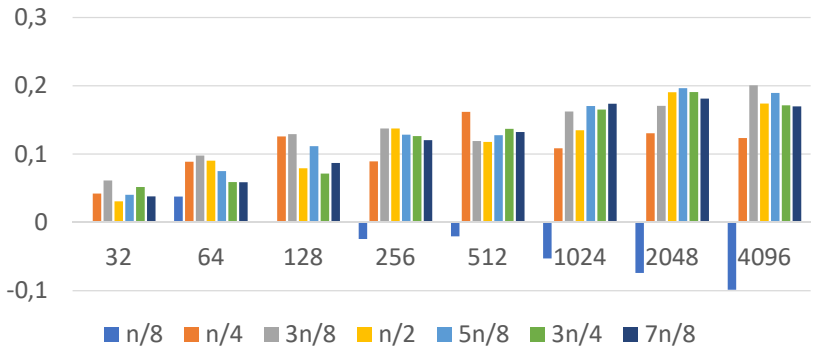


Figura 4.11: gap di miglioramento di GREEDY rispetto a BLUM sul primo dataset.

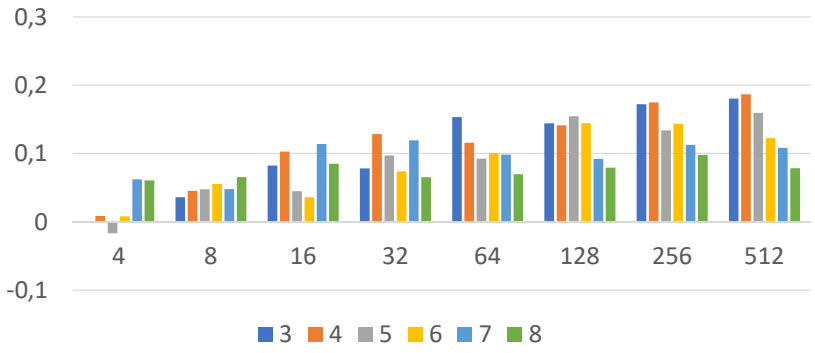


Figura 4.12: gap di miglioramento di GREEDY rispetto a BLUM sul secondo dataset.

4.9 Comparazione con $\mathcal{A}3$

Adesso si vogliono valutare l'euristiche definite con un approccio un po' diverso da quello utilizzato per le analisi precedenti. Nello specifico, si effettua un'analisi comparativa dei metodi progettati con un particolare algoritmo di approssimazione, descritto nella sezione 2.3.1 con il nome $\mathcal{A}3$. Per questo algoritmo randomizzato, si è mostrato un limite superiore per il rapporto di prestazione e si è anche fornita una famiglia di istanze dove l'algoritmo esibisce il comportamento pessimo. In particolare, il valore limite è variabile e corrisponde ad $m = \max_{\sigma \in \Sigma} \min\{f(x,\sigma), f(y,\sigma)\}$, per una generica istanza (x, y) ed un alfabeto Σ. Formalmente significa che $|\mathcal{A}3(x,y)| \geq \frac{|\omega^*|}{m}$, dove ω^* è una soluzione ottima.

La famiglia di istanze è quindi un'insieme $\mathcal{F}$ tale che $\forall (x,y) \in \mathcal{F}, |\mathcal{A}3(x,y)| = \frac{|\omega^*|}{m}$. Si riporta qui il pattern che caratterizza la famiglia, mentre si rimanda alla sezione 2.3.1 per la dimostrazione. Dunque, per un alfabeto $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$, si definisce una stringa $x = \sigma_1\sigma_2\cdots\sigma_k$, le istanze sono della forma $(x^m, \sigma_k^{m+m_k}\sigma_{k-1}^{m+m_{k-1}}\cdots\sigma_1^{m+m_1})$, con $m, m_1, m_2, \ldots, m_k \in \mathbb{N}^+$.

La soluzione ottima ha lunghezza m, mentre $\mathcal{A}3$ trova una soluzione composta da un solo elemento. Per analizzare il grado di approssimazione dell'algoritmo, la famiglia definita è sufficiente. D'altro canto risulta meno interessante, dal momento in cui si vuole confrontare l'algoritmo approssimato con un'euristica. Per questo motivo si presenta una generalizzazione di $\mathcal{F}$ che permette una lunghezza arbitraria per la soluzione ritornata da $\mathcal{A}3$. Ci si pone dunque l'obiettivo di trovare una generalizzazione che permetta di definire delle istanze con soluzione ottima arbitrariamente più grande di m o, più formalmente, $|\omega^*| = n \cdot m$ con $n \in \mathbb{N}^+$.

Si osservi che, costruendo delle istanze definite su alfabeti disgiunti, è possibile concatenare queste istanze ottenendo una nuova macro-istanza, dove i pattern restano intatti, ciò è vero in quanto, l'insieme dei *cross* che si viene a creare, altro non è che l'unione dei cross dei singoli pattern, e quindi è possibile operare sulle singole istanze in maniera indipendente e unire le singole soluzioni, con la garanzia che la loro concatenazione è ammissibile per la macro-istanza. In generale, si definiscono n alfabeti

distinti $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ tali che $\Sigma_i \cap \Sigma_j = \emptyset \ \forall i, j \in \{1, 2, \ldots, n\}, i \neq j$. La prima stringa sarà quindi la concatenazione di stringhe della forma

$$x_i = \sigma_{i,1}\sigma_{i,2} \cdots \sigma_{i,k_i}, \forall i \in \{1, 2, \ldots, n\}$$

con $k_i = |\Sigma_i|$. La seconda stringa si ottiene concatenando stringhe

$$y_i = \sigma_{i,k_i}^{m+m_{i,k_i}} \sigma_{i,k_i-1}^{m+m_{i,k_i-1}} \cdots \sigma_{i,1}^{m+m_{i,1}}, \forall i \in \{1, 2, \ldots, n\}.$$

L'istanza completa è quindi la seguente: $(x_1^m x_2^m \cdots x_n^m, y_1 y_2 \cdots y_n)$. Una soluzione ottima in questo caso corrisponderà alla concatenazione ordinata di soluzioni ottime per ogni (x_i^m, y_i). Allo stesso modo la soluzione trovata da $\mathcal{A}3$ sarà una concatenazione ordinata di simboli, scegliendo un qualsiasi $\sigma \in \Sigma_i, \forall i \in \{1, 2, \ldots, n\}$. La lunghezza sarà quindi di esattamente n simboli mentre la soluzione ottima è pari ad $n \cdot m$, come voluto.

A questo punto si effettua un ulteriore passo di generalizzazione allo scopo di rendere più complessa l'individuazione del pattern. Si procede quindi con l'inserimento di simboli che fungno da rumore, senza però andare a modificare soluzione ottima e soluzione approssimata. Per fare ciò si definiscono due alfabeti disgiunti, detti Σ_x e Σ_y. Occorre scegliere degli alfabeti nuovi, ossia tali che $\Sigma_x \cap \Sigma_i = \emptyset, \Sigma_y \cap \Sigma_i = \emptyset$ $\forall i \in \{1, 2, \ldots, n\}$.

La nuova famiglia sarà composta da istanze in cui è possibile individuare una sotto-istanza che verifichi il pattern precedente. Di fatto è possibile mantenere le propretà del pattern conservandone l'ordinamento, mentre non è necessario che gli elementi siano contigui. Questi possono infatti alternarsi ad elementi dei nuovi alfabeti, utilizzando Σ_x per la prima stringa e Σ_y per la seconda. Utilizzando due alfabeti così definiti, si possono ottenere delle istanze in cui il pattern risulta sparso e più difficile da individuare.

Anche in questo caso il pattern è garantito dal fatto che il numero di cross ottenuto dal passo di generalizzazione non è aumentato. Siccome si sono scelti due alfabeti nuovi e disgiunti, questi non apporteranno alcun match aggiuntivo, di conseguenza l'insieme delle possibili soluzioni resta invariato.

Metodo di Campionamento

La costruzione di un campione casuale per la famiglia descritta avviene secondo un approccio ispirato al *planted random model* [17], che consiste nell'inserire all'interno di un'istanza una soluzione nota. Il metodo originale agisce in questo modo: fissato un $l \leq |\Sigma|$, scegliere una sequenza ω repetition-free di lunghezza l definita su Σ. Successivamente, generare una sequenza $\overline{x}$ di $\overline{n}$ simboli in Σ in modo casuale, uniforme e indipendente. Infine, scegliere casualmente un'insieme $\{s_1, \ldots, s_l\} \subseteq \{1, \ldots, \overline{n}\}$ di posizioni distinte di $\overline{x}$ e sostituire ogni simbolo s_i di $\overline{x}$ con il simbolo i-esimo di ω. Sia x la sequenza così ottenuta. Si ripeta la procedura in modo analogo per una seconda sequenza $\overline{y}$ di $\overline{n}$ simboli scelti a caso ma con la stessa sequenza ω, ottenendo una nuova sequenza y. Le sequenze risultanti x e y sono tali che $\texttt{OPT}(\texttt{RFLCS}(x, y)) \geq l$.

Si procede in maniera simile per il campionamento della famiglia. La differenza è che invece di costruire una sequenza repetition-free ω si costruiscono due sequenze $x_1^m x_2^m \cdots x_n^m$ e $y_1 y_2 \cdots y_n$ da "piantare" nelle stringhe $\overline{x}$ ed $\overline{y}$, definite rispettivamente sugli alfabeti Σ_x e Σ_y. L'inserimento delle due sequenze all'interno di $\overline{x}$ e $\overline{y}$ avviene in maniera analoga al metodo originale.

Impostazione Analisi

Fin ora si è descritto nel dettaglio la famiglia di istanze su cui basare la valutazione degli algoritmi. Adesso si vuole entrare nel merito del potenziale valore aggiunto dalla risoluzione di un campione appartenente a questa famiglia. Grazie alla definizione di questo pattern è possibile acquisire un campione di istanze, di una taglia tale da rendere insostenibile il calcolo di una soluzione ottima, ma il cui valore di funzione obiettivo e noto a priori. Inoltre si è dimostrato che il pattern scelto porta anche l'algoritmo approssimato $\mathcal{A}3$ ad esibire il suo comportamento peggiore e anche in questo caso non è necessario effettuare alcun esperimento.

Si propone quindi un metodo di comparazione alternativo di una certa euristica $\mathcal{H}$. La bontà di $\mathcal{H}$ è valutata tramite calcolo dei valori di funzione obiettivo e confronto con soluzione ottima e soluzione approssimata.

Formalmente, si riprende il simbolo m per indicare il fattore di approssimazione nonché lunghezza della soluzione calcolata da $\mathcal{A}3$, mentre nm è il valore di funzione obiettivo ottimo per un'istanza $(x, y) \in \mathcal{F}$. Ci si chiede se e come si colloca $|\mathcal{H}(x, y)|$ all'interno dell'intervallo $[m, nm]$.

Il dataset creato è formato da istanze (x, y) che verificano $|\omega^*| = \frac{\bar{n}}{c}$ ed $m = 3$, per ω^* soluzione ottima e $\bar{n} = |x| = |y|$. Si sono scelte 6 dimensioni diverse e per ogni dimensione sono presenti 10 istanze. Le soluzioni approssimate ritornate da $\mathcal{A}3$ hanno lunghezza pari ad $n \in \{24, 32, 48, 64, 96, 128\}$ e quindi $|\omega^*| \in \{72, 96, 144, 192, 288, 384\}$. Fissato $c = 10.\bar{6}$, si ottengono le rispettive taglie $\bar{n} \in \{768, 1024, 1536, 2048, 3072, 4096\}$.

Risultati

In prima analisi si studia il comportamento dell'algoritmo GS sul data-set generato. Dal grafico a dispersione in figura 4.13 si osserva un comportamento quasi lineare. Seppur di poco, il gap dalla soluzione ottima è in realta crescente, all'aumentare della taglia. Tuttavia sembra stabilizzarsi sul 13% laddove il gap della soluzione approssimata è fisso al 67%.

Per quanto riguarda le meta-euristiche ILS e GRASP, i risultati sono ben diversi. Entrambi gli algoritmi hanno infatti ottenuto le soluzioni ottime per ogni istanza. La motivazione alla base di questo successo è dovuto a due componenti che accomunano i due algoritmi. Nello specifico, ci si riferisce alla componente di costruzione di una soluzione ed alla componente di ricerca locale, entrambe basate su criteri euristici adeguati alla famiglia di istanze.

Soffermandosi sulla famiglia con un solo passo di generalizzazione (quindi senza aggiunta del rumore), si può notare come il criterio greedy crei, per ogni simbolo, un ordinamento totale sui candidati che vede come candidato più appetibile proprio quello appartenente ad una soluzione ottima. Sottoponendo il data-set all'euristica in questione, si evince una legge lineare che separa la soluzione GREEDY dall'ottimo (figura 4.14). Tale gap costante si ritiene essere un errore causato dall'ultimo passo di generalizzazione della famiglia, il quale introduce un rumore che ha proprio lo scopo di annichilire ogni relazione tra bontà di una copppia e relativa distanza. Tale errore viene

infine coperto dal metodo di ricerca locale che riesce a chiudere il gap, raggiungendo la soluzione ottima. Si ricorda che l'intuizione da cui parte la ricerca locale è che, per una qualsiasi stringa ω, sottosequenza di (x, y), possono occorrere diverse sottosequenze in x e y corrispondenti alla stessa stringa ω, e quindi, a partire dalla stringa ω è possibile trovare una soluzione ammissibile $\bar{\omega}$ tale che $|\bar{\omega}| > |\omega|$, provando ad aggiungere i simboli mancanti (non presenti in ω). Per la famiglia definita questo fenomeno si presenta molto spesso e quindi la ricerca locale è estremamente efficace.

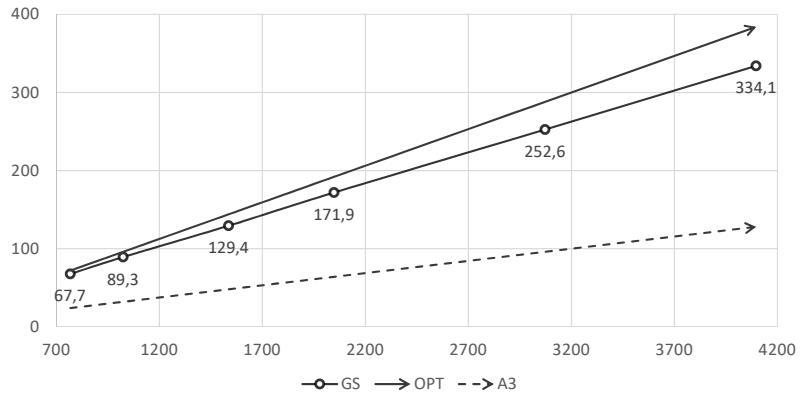


Figura 4.13: comparazione GS con valori ottimi (OPT) e approssimati (A3).

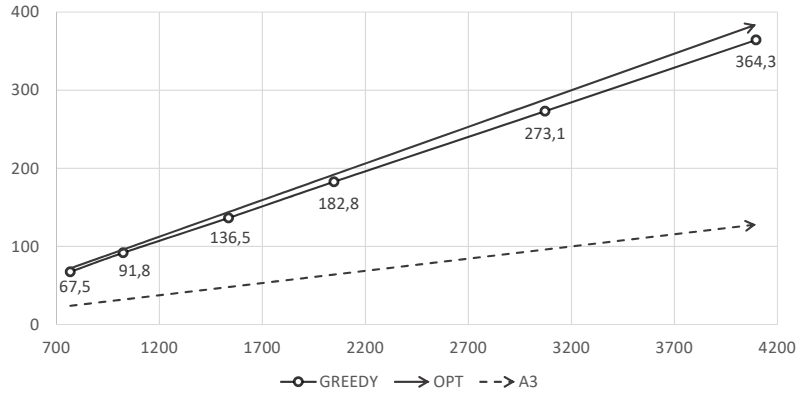


Figura 4.14: comparazione GREEDY con valori ottimi e approssimati.

Capitolo 5

Conlcusioni

Con questa tesi si è sviluppata un'indagine approfondita della metrica Repetition Free Longest Common Subsequence. Indagine che parte da un'introduzione motivazionale, in cui si colloca l'argomento nel panorama della Bioinformatica. Si passa in seguito ad una ricerca dei risultati teorici presenti in letteratura, da cui è emersa l'intrattabilità del calcolo esatto. Per questo motivo si è affrontato il problema tramite tecniche di ottimizzazione euristiche allo scopo di calcolare delle soluzioni di buona qualità in un ragionevole lasso di tempo. Si sono quindi presentate 3 diverse meta-euristiche ed un criterio di costruzione basato su valutazioni *greedy*. Gli algoritmi sono stati messi a confronto con lo stato dell'arte, allo scopo di valutarne la validità scientifica. I risultati ottenuti sono soddisfacenti da diversi punti di vista. Nello specifico, l'algoritmo GRASP ha mostrato valori migliori della controparte su istanze di taglia piccola. Adottare una nuova funzione greedy si è dimostrata una scelta vincente, portando a risultati che da soli hanno potuto competere con un'intera metaeuristica, pubblicata nel 2013. Infine si è osservato graficamente, tramite calcolo del gap di miglioramento, che gli algoritmi qui proposti ottengono quasi ovunque dei risultati che reggono il confronto con l'attuale stato dell'arte, con gap medio al di sotto del 2%. A dare ulteriore valenza ai nuovi algoritmi ci sono le prestazioni in termini di running-time. I tempi di calcolo, necessari a raggiungere la convergenza, sono infatti estremamente bassi, mai superiori ad un numero di secondi pari a $2(l_x + l_y - |\Sigma|) \cdot 10^{-3}$. In questo senso quindi si può affermare che, gli algoritmi qui proposti, offrono delle soluzioni dalla qualità competitiva e con

dei tempi di attesa estremamente bassi. Caratteristiche del genere sono un requisito comune in contesti applicativi reali, rendendo GRASP e GS una valida alternativa ad Hyb-CMSA.

5.1 Sviluppi Futuri

Durante l'attività di tesi si è appreso un quadro generale dei progressi della ricerca nello studio del problema Repetition Free Longest Common Subsequence. Contemporaneamente si è arrivati anche allo sviluppo di possibili spunti di lavoro da intraprendere in futuro. Primo tra tutti si vuole migliorare lo stato dell'arte per quanto riguarda le tecniche di ottimizzazione implementate per la risoluzione del problema. Oltre all'implementazione di ulteriori framework, potrebbe essere un'ipotesi vincente quella di introdurre la costruzione GREEDY qui presentata nell'algoritmo Hyb-CMSA, come alternativa al metodo di costruzione probabilistico originale. Un confronto preliminare tra i due metodi potrebbe dimostrare la superiorità dell'algoritmo GREEDY in una valutazione indipendente dall'intero algoritmo, alla quale seguirebbe una comparazione completa.

Per quanto riguarda gli algoritmi proposti in questa tesi, essi offrono ancora dei margini di miglioramento. GRASP, ILS e GS utilizzano una componente di risoluzione per il problema LCS. Questo porta a svariate opportunità di miglioramento rese possibili dai numerosi studi dedicati al calcolo di una LCS. Dati i tempi estremamente competitivi impiegati dai tre algoritmi, ci si è soffermati all'implementazione maggiormente nota, utile come proof-of-concept. Esiste però un'alternativa che potrebbe risultare mediamente più efficiente. Trattasi di un algoritmo con complessità $\mathcal{O}((r+n)\log n)$, dove r è il numero totale di coppie ordinate (i,j) tali che $x_i = y_j$, per un'istanza (x,y) [29]. Quindi nel caso peggiore si ha una complessità pari a $\mathcal{O}(n^2 \log n)$. Tuttavia, nelle tipologie di istanza in cui la maggior parte delle posizioni di una sequenza corrisponde relativamente a poche posizioni nell'altra sequenza, è possibile prevedere un tempo di esecuzione di $\mathcal{O}(n \log n)$. Si pensi ad esempio alle istanze con $|\Sigma| = 7n/8$.

Potrebbe inoltre essere opportuno analizzare il comportamento degli algoritmi di

ottimizzazione, già esistenti o nuovi progettati ad hoc, in contesti maggiormente applicativi, tramite ad esempio l'uso di dati, con una pertinenza biologica, provenienti dal sequenziamento.

Infine sarebbe interessante studiare l'attendibilità della metrica RFLCS. Ad oggi sembra mancare un filone che cerchi di valutare l'utilità di RFLCS in contesti reali. Alcuni spunti sono reperibili in [30], articolo nel quale si propone un metodo di comparazione tra diverse metriche che permette di analizzarne utilità e robustezza.

Appendice A

Risultati Sperimentali

A.1 GRASP

Parametri

Prima di sperimentare l'algoritmo sull'intero data set, occorre stabile che valore dare ai parametri di cui necessità.

In questo caso si parla di due parametri, entrambi correlati alla distribuzione di probabilità utile alla costruzione greedy randomizzata, detta $\mathcal{D}$. I parametri in questione quindi definiscono il metodo con cui calcolare le distribuzioni di probabilità da associare ad una lista di candidati. In questo caso, i candidati saranno le occorrenze presenti nell'istanza. Ogni letterà dell'alfabeto avrà una lista di candidati dedicata, in cui compariranno tutte le relative occorrenze, a prescindere dalla stringa di appartenenza. Si ricorda che la lista è ordinata coerentemente con i valori greedy associati ai candidati. A questo punto occorre capire se è più opportuno basare la distribuzione semplicemente sull'ordinamento, oppure se dare ulteriore peso al criterio greedy, basando i valori di $\mathcal{D}$ anche su quest'ultimo. Formalmente, sia $\mathcal{L}$ una lista di occorrenze per una lettera $a \in \Sigma$. L'elemento i-esimo di $\mathcal{L}$ è detto e_i. Il valore greedy associato all'elemento sarà $g(e_i)$. Si definisce quindi il calcolo della probabilità empirica nei seguenti due modi:

- $\mathbb{P}(e_i) = \frac{g(e_i)}{\sum\limits_{e_j \in \mathcal{L}} g(e_j)}$

- $\mathbb{P}(e_i) = \frac{i}{\sum_{e_j \in \mathcal{L}} j}$

Nell'implementazione finale si è però aggiunto un'ulteriore passo di calcolo. Si è infatti prevista la possibilità di applicare una funzione di distorsione (*biasing*) al valore associato ad un elemento, indipendentemente da se è stato ottenuto tramite funzione greedy o ordinamento. Si generalizza quindi ad una funzione $h : \mathcal{L} \to \mathbb{R}$ (*heuristic*). In [23] introducono la seguente equazione:

$$\mathbb{P}(e_i) = \frac{\text{bias}(h(e_i))}{\sum_{e_j \in \mathcal{L}} \text{bias}(h(e_j))}$$

Il metodo di calcolo sarà dunque parametrizzato in maniera bi-dimensionale. Il primo parametro descritto avrà dominio codificato come l'insieme {GreedyBased, OrderBased}. Mentre il dominio della funzione di *biasing* è descritto nella tabella A.1

Distribuzione	Nome	Funzione
Esponenziale	exp	e^{-x}
Logaritmica	log	$\log(x+2)^{-1}$
Polinomiale	p	$(x+1)^{-1}$
Uniforme	$\mathcal{U}$	c
Identità	$\mathcal{I}$	x^{-1}

Tabella A.1: funzioni di *biasing* usate per pesare in vari modi le liste ordinate di elementi candidati.

Si osservi che i bias proposti vedono il valore di input al denominatore. Tale operazione è necessaria perchè occorre che le probabilità decrescano al crescere del valore greedy, in quanto il criterio associa valori bassi ad elementi considerati buoni.
Anche in questo caso, capire quale sia l'operazione migliore da effettuare, non è semplice e dipende molto dall'uso che ne fa l'intero algoritmo. Un bias esponenziale offre una bassa diversificazione tra diverse generazioni, ma può risultare adeguata per la costruzione di una buona soluzione di partenza. Al contrario, se il criterio greedy fosse inattendibile, sperimentalmente ci si aspetterebbe che $bias \sim \mathcal{U}$.

Budget

L'ultima decisione da prendere prima di procedere con il tuning è il budget da dare all'algoritmo. Come già scritto nel capitolo di presentazione dell'algoritmo (3.1), il budget assegnato dovrà essere un tempo limite. Per capire quale fosse un tempo appropriato si è proceduto tramite una sperimentazione preliminare, da cui si è cercato di capire di quanto budget necessita GRASP per convergere al miglior valore che è in grado di ottenere. Per una versione del data set ridotta del 95% (trattasi di 200 istanze), si è osservato che un budget pari a $(l_x + l_y) + (l_x + l_y)(1 - \frac{2|\Sigma|}{l_x + l_y})$ millisecondi è sufficiente a raggiungere la convergenza. L'idea alla base di questa espressione è di dare un budget comune alle istanze della stessa taglia (primo addendo), al quale aggiungere un "bonus" che cresce con il divario tra taglia e alfabeto. Equivalentemente il budget dell'algoritmo sarà di $2(l_x + l_y - |\Sigma|) \cdot 10^{-3}$ secondi. Si osservi che questo calcolo, oltre a dare un budget crescente al crescere della taglia, sarà crescente al diminuire del rapporto tra cardinalità dell'alfabeto e taglia. In effetti un rapporto basso porta ad un numero alto di possibili soluzioni, rendendo più complessa la ricerca.

Configurazione

Per la configurazione si è utilizzato il framework irace, descritto nella sezione 4.4. Il tuning prevede due fasi distinte. Si otterranno quindi due diverse combinazioni di parametri, da utilizzare per le due diverse macro-tipologie di istanze per ognuna delle quali si è generato un training set apposito, seguendo il metodo descritto nella sezione 4.3.

Per il primo set si è ottenuta la configurazione OrderBased, log. Mentre per il secondo set il tuner suggerisce la configurazione GreedyBased, $\mathcal{I}$.

Risultati

A seguire si riportano i valori di funzione obiettivo ottenuti tramite esecuzione dell'algoritmo GRASP sull'intero data-set. I risultati sono presentati in forma tabellare. Per il primo insieme di dati si faccia riferimento alla tabella A.2, in cui sono riportati i valori di funzione obiettivo ottenuti. Ogni colonna rappresenta una tipologia d'istanza

distinta dalla dimensione dell'alfabeto in funzione della taglia dell'istanza. Guardando ad una sola riga si osservano i risultati per tutte le tipologie, ma limitate ad una sola taglia n.

$\frac{\lvert\Sigma\rvert}{n}$	$n/8$	$n/4$	$3n/8$	$n/2$	$5n/8$	$3n/4$	$7n/8$
32	4	7,83	8,77	8,87	8,60	8,17	7,67
64	8	14,67	15,53	14,80	13,30	12,53	11,57
128	16	25,77	24,83	22,93	21,20	19,70	18,40
256	31,97	42,33	39,43	35,07	32,43	29,97	27,80
512	60,20	63,20	57,67	52,03	47,67	44,43	40,57
1024	99,33	94,17	85,43	76,43	69,13	64,57	60,37
2048	152,67	136,17	121,20	109,33	99,97	92,73	87,20
4096	228,80	194,73	172,80	157,03	143,17	132,83	124,83

Tabella A.2: Risultati esecuzione algoritmo GRASP sul primo insieme di dati.

L'esito per il secondo data set è riportato in tabella A.3. Qui le colonne rappresentano le tipologie di istanze, che si distinguono in base al numero massimo di ripetizioni consentite. Mentre le righe corrispondo ad istanze della stessa taglia dell'alfabeto. Si ricorda che per f_{max} fissato, il rapporto tra cardinalità dell'alfabeto e valore atteso della lunghezza delle sequenze è costante.

f_{max} / $\lvert\Sigma\rvert$	3	4	5	6	7	8
4	3,47	3,77	3,83	3,90	3,97	3,97
8	6,23	6,87	7,40	7,53	7,70	7,77
16	9,70	11,57	12,93	14,00	14,93	14,80
32	16,13	19,00	21,60	23,73	25,40	27,37
64	25,43	30,27	34,57	38,13	42,47	43,73
128	36,77	44,27	51,67	58,17	63,80	68,10
256	54,40	66,33	75,97	85,40	94,90	101,87
512	79,87	95,03	110,63	123,37	136,77	148,43

Tabella A.3: Risultati esecuzione algoritmo GRASP sul secondo insieme di dati.

A.2 ILS

Configurazione Parametri e Budget

I parametri richiesti dall'Iterate Local Search sono tre. Due di questi sono gli stessi necessari al GRASP per la costruzione di una soluzione, la loro descrizione è quindi riportata nella sotto-sezione A.1. In questo caso i parametri serviranno alla costruzione di una soluzione di partenza, dopo la quale si avvierà la fase composta da ricerca locale e diversificazione. In paricolare, il terzo parametro servirà durante la diversificazione. Si ricorda che questa componente a partire da una soluzione vi applica una perturbazione per eliminazione. Sceglierà in maniera casuale degli elementi in soluzione, cancellandoli. Occorre però capire quante eliminazioni effettuare. Si pensa dunque che questo valore debba essere in funzione della taglia della soluzione in esame. Si definisce quindi un fattore $d \in [0, 1]$ che esprime la percentuale di elementi da eliminare dalla soluzione. Il tuner irace considera, come configurazione elite, un approccio HeuristicBased, con bias $= \mathcal{I}$ e $d = 0,35$ nel caso del primo insieme, mentre per il secondo data set propone OrderBased, log, 0.15.

Il budget assegnato è lo stesso tempo necessario al GRASP per raggiungere la convergenza, ovvero $2(l_x + l_y - |\Sigma|) \cdot 10^{-3}$ secondi.

Risultati

Da una prima occhiata delle tabelle A.4 e A.5 appare abbastanza evidente che l'algoritmo non riesce a raggiungere GRASP, ottendo ovunque dei valori più bassi. Occorre però quantificare questo distacco. Un'analisi accurata è reperibile alla sezione 4.6.

$\frac{\lvert\Sigma\rvert}{n}$	$n/8$	$n/4$	$3n/8$	$n/2$	$5n/8$	$3n/4$	$7n/8$
32	4	7,8	8,7	8,7	8,37	8	7,53
64	8	14,57	15,47	14,63	12,8	12,1	11,1
128	16	25,1	23,73	21,87	20,47	18,9	17,5
256	31,97	40,33	36,73	32,13	30,07	28,33	26,27
512	61,37	58,87	52,6	48,03	43,6	41,1	38,1
1024	101,1	86,33	78,17	69,63	64,33	59,73	56,37
2048	152,73	124,93	111,37	102,03	93,73	87,53	82,6
4096	222,27	180,37	160,07	146,4	135,67	125,87	118,33

Tabella A.4: Risultati esecuzione algoritmo ILS sul primo insieme di dati.

f_{max} / $\lvert\Sigma\rvert$	3	4	5	6	7	8
4	3,3	3,7	3,73	3,83	3,93	3,93
8	5,9	6,67	7,07	7,37	7,57	7,73
16	9	10,97	12,2	13,73	14,8	14,43
32	14,87	17,5	20,5	22,63	24,47	26,5
64	23,3	27,4	31,73	34,97	39,07	41,57
128	33,83	39,67	46,3	52,6	57,43	63,23
256	49,8	59,8	69	77,23	85,07	92,63
512	73,9	86,17	101,43	111,83	124,37	136,43

Tabella A.5: Risultati esecuzione algoritmo ILS sul secodo insieme di dati.

A.3 GS

Configurazione Parametri e Budget

L'euristica Generate and Solve prevede cinque parametri, due dei quali sono già stati utilizzati per GRASP e ILS. Trattasi quindi dei parametri utili al calcolo della distribuzione di probabilità empirica presente durante la generazione di una sotto-istanza. I restanti valori da configurare servono alle componenti di feedback, quali τ ed age_v.
Per quanto riguarda τ si vuole definire un metodo che adatti il valore $\tau(\sigma)$, per un generico $\sigma \in \Sigma$, al numero di sue ripetizioni all'interno della soluzione non ammissibile s creata nel primo step del risolutore. Si ricorda che la componente di risoluzione risolve il problema rilassato LCS e successivamente corregge l'inammissibilità, eliminando le ripetizioni. In una generica sequenza s, fissata una lettera $\sigma \in \Sigma$, possono verificarsi due casi, per i quali si procede con due operazioni differenti. L'aggiornamento è descritto dalla seguente equazione:

$$\tau(\sigma) \leftarrow \begin{cases} \tau(\sigma) + \tau(\sigma)\epsilon^{-} & \text{se } f(s,\sigma) > 1 \\ \tau(\sigma) + \tau(\sigma)(1-\epsilon^{+}) & \text{altrimenti.} \end{cases} \tag{A.1}$$

Il primo caso si verifica se la soluzione s, in generale non ammissibile, presenta delle ripetizioni di σ In questo caso il valore di $\tau(\sigma)$ dovrà diminuire di una quantità proporzionale a $\epsilon^- \in [0, 1]$. Al contrario se non vi sono ripetizioni in s si vuole incrementare il valore di $\tau(\sigma)$ proporzionalmente a $\epsilon^+ \in [-1, 0]$. L'ultimo parametro riguarda la funzione $age_v : \mathbb{N} \to \mathbb{N}$. Si introduce quindi l'età massima A_{max}. Quando un elemento viene inserito nella sotto-istanza ma non compare in soluzione, la sua età aumenta, e se raggiunge A_{max} viene eliminato dall'istanza. Per la configurazione di A_{max} si sono scelti i valori $\{1, 2, 3, 4, 5, 7, 10, \infty\}$. Il tuning ha portato alle seguenti configurazioni:

- set 1 - GreedyBased, bias = $\mathcal{I}$, $A_{max} = 7$, $\epsilon^+ = 0.4$, $\epsilon^- = -0.25$
- set 2 - GreedyBased, bias = p, $A_{max} = 5$, $\epsilon^+ = 0.2$, $\epsilon^- = -0.1$

In ultima analisi, si è fissato come budget lo stesso tempo limite usato per ILS e GRASP, ossia $2(l_x + l_y - |\Sigma|) \cdot 10^{-3}$ secondi.

Risultati

$\frac{\lvert\Sigma\rvert}{n}$	$n/8$	$n/4$	$3n/8$	$n/2$	$5n/8$	$3n/4$	$7n/8$
32	4	7,83	8,73	8,83	8,53	8,1	7,57
64	8	14,33	15,2	14,53	13,07	12,43	11,5
128	16	24,63	24,13	22,3	20,83	19,57	18,27
256	31,9	41,37	38,67	34,37	32,2	29,73	27,43
512	60,13	64	57,73	51,7	47,3	44	40,07
1024	103,5	98,87	88,07	77,23	69,33	64,23	60,2
2048	167,93	149,13	127,8	113,63	102,73	93,7	87,6
4096	266,7	222,57	189,43	166,03	149	135,8	126,9

Tabella A.6: Risultati esecuzione algoritmo GS sul primo insieme di dati.

f_{max} / $\|\Sigma\|$	3	4	5	6	7	8
4	3,43	3,77	3,83	3,9	3,97	3,97
8	6,17	6,83	7,3	7,47	7,67	7,63
16	9,67	11,33	12,6	13,57	14,47	14,23
32	15,97	18,57	20,73	22,77	24,13	25,93
64	25,07	29,23	33,47	37	40,67	42,53
128	36,4	43,63	50,97	58,2	63,47	68,03
256	54,77	67,47	78,43	88,43	98,6	106,83
512	81,13	99,23	117,37	133,37	148,27	163,37

Tabella A.7: Risultati esecuzione algoritmo GS sul secodo insieme di dati.

A.4 GREEDY

$\|\Sigma\|$ / n	$n/8$	$n/4$	$3n/8$	$n/2$	$5n/8$	$3n/4$	$7n/8$
32	4	7,13	7,67	7,5	7,43	7,17	6,83
64	7,93	12,4	12,97	12,5	11,6	11,37	10,23
128	14,43	20,13	19,9	18,93	18,2	17,23	16,43
256	25,07	31,37	31,07	28,83	28,57	26,13	24,67
512	39,4	48,23	46,73	43,67	41,87	38,9	36,33
1024	61,47	71,07	71,5	64,87	60,7	57,17	54,37
2048	91,33	105,33	102,6	97,63	90,4	83,77	79,27
4096	136,03	153,47	150	140,77	131,77	122,37	116,03

Tabella A.8: Risultati ottenuti dall'esecuzione del modulo di costruzione di una soluzione per il primo insieme di dati.

f_{max} / $\lvert\Sigma\rvert$	3	4	5	6	7	8
4	3	3,5	3,47	3,73	3,87	3,8
8	5,57	5,97	6,3	6,7	6,9	7,17
16	8,5	9,73	10,5	11,17	12,3	12,13
32	14,07	15,57	16,8	18,57	19,53	20,43
64	22	24,73	26,63	29,43	31,5	32
128	32,1	36,8	41,2	44,53	47,03	49,27
256	47,47	55,77	61,53	66,73	71,37	74,27
512	70,93	82,57	91,33	98,07	104,43	110,17

Tabella A.9: Risultati ottenuti dall'esecuzione del modulo di costruzione di una soluzione per il secondo insieme di dati.

A.5 BLUM

$\lvert\Sigma\rvert$ / n	$n/8$	$n/4$	$3n/8$	$n/2$	$5n/8$	$3n/4$	$7n/8$
32	4	6,83	7,2	7,27	7,13	6,8	6,57
64	7,63	11,3	11,7	11,37	10,73	10,7	9,63
128	14,43	17,6	17,33	17,43	16,17	16	15
256	25,7	28,57	26,8	24,87	24,9	22,83	21,7
512	40,23	40,43	41,17	38,53	36,53	33,57	31,53
1024	64,9	63,37	59,9	56,13	50,37	47,73	44,93
2048	98,63	91,6	85,1	79,03	72,63	67,8	64,9
4096	150,93	134,53	119,9	116,3	106,8	101,4	96,33

Tabella A.10: risultati euristica di costruzione utilizzata in Hyb-CMSA sul primo data set.

f_{max} / $\lvert\Sigma\rvert$	3	4	5	6	7	8
4	3	3,47	3,53	3,7	3,63	3,57
8	5,37	5,7	6	6,33	6,57	6,7
16	7,8	8,73	10,03	10,77	10,9	11,1
32	12,97	13,57	15,17	17,2	17,2	19,1
64	18,63	21,87	24,17	26,47	28,4	29,77
128	27,47	31,6	34,83	38,1	42,7	45,37
256	39,3	46,03	53,3	57,17	63,33	67
512	58,13	67,17	76,77	86,07	93,13	101,53

Tabella A.11: risultati euristica di costruzione utilizzata in Hyb-CMSA sul secondo data set.

A.6 EA

$\lvert\Sigma\rvert$ / n	$n/8$	$n/4$	$3n/8$	$n/2$	$5n/8$	$3n/4$	$7n/8$
32	4,00	7,77	8,70	8,53	8,23	7,93	7,53
64	8,00	14,07	15,33	14,10	12,40	12,37	10,97
128	16,00	24,27	24,37	22,00	20,33	19,53	17,70
256	31,83	40,97	38,20	33,10	31,33	29,00	26,93
512	58,33	59,63	48,77	46,50	43,47	39,47	37,57
1024	86,00	73,47	63,83	61,30	56,00	51,90	50,30
2048	107,10	91,97	79,43	76,87	72,83	67,40	65,83
4096	129,43	117,57	107,67	102,10	96,43	90,70	86,03

Tabella A.12: Risultati esecuzione algoritmo EA sul primo insieme di dati.

f_{max} / $\lvert\Sigma\rvert$	3	4	5	6	7	8
4	3,47	3,77	3,83	3,90	3,97	3,97
8	6,23	6,83	7,33	7,50	7,63	7,77
16	9,60	11,47	12,80	13,93	14,80	14,57
32	16,03	18,80	21,23	23,37	24,83	26,63
64	25,10	30,00	34,30	37,73	41,23	42,23
128	36,00	41,83	48,90	55,1	60,8	66,00
256	49,87	57,27	63,10	70,3	75,67	81,20
512	66,17	74,57	80,47	86,73	92,73	100,23

Tabella A.13: Risultati esecuzione algoritmo EA sul secodo insieme di dati.

A.7 Hyb-CMSA

$\lvert\Sigma\rvert$ / n	$n/8$	$n/4$	$3n/8$	$n/2$	$5n/8$	$3n/4$	$7n/8$
32	4	7,83	8,77	8,87	8,57	8,17	7,67
64	8	14,63	15,53	14,77	13,30	12,53	11,57
128	16	25,77	24,87	22,90	21,20	19,70	18,40
256	31,97	43,63	39,97	35,10	32,50	29,97	27,80
512	63,13	67,90	59,77	53,10	47,83	44,53	40,57
1024	111,17	103,00	90,50	79,03	70,03	65,07	60,50
2048	180,50	154,33	130,57	115,30	103,80	94,53	88,00
4096	279,17	226,67	191,37	167,47	150,00	136,57	127,20

Tabella A.14: risultati di Hyb-CMSA sul primo data set.

f_{max} / $\lvert\Sigma\rvert$	3	4	5	6	7	8
4	3,47	3,77	3,83	3,9	3,97	3,97
8	6,23	6,83	7,4	7,53	7,7	7,77
16	9,7	11,57	12,93	13,87	14,87	14,77
32	16,13	19	21,63	23,7	25,53	27,4
64	25,43	30,37	34,87	39,07	43,5	45,17
128	36,7	44,9	53,23	61,07	67,9	73,57
256	54,97	68,7	81	93,1	103,5	113,7
512	81,57	100,83	120,43	137,03	154,57	172,1

Tabella A.15: risultati di Hyb-CMSA sul primo data set.

Riferimenti

[1] S. S. Adi et al. *Repetition-free longest common subsequence* (cit. alle pp. ii, iii, 9, 12, 18, 19, 48, 51).

[2] *Rosalind - Problems*. URL: `http://rosalind.info/problems/list-view` (cit. a p. 1).

[3] S. B. Needleman e C. D. Wunsch. *A general method applicable to the search for similarities in the amino acid sequence of two proteins* (cit. a p. 3).

[4] J. Kleinberg e É. Tardos. *Algorithm Design* (cit. a p. 3).

[5] C. J. Harrison e J. A. Langdale. *A step by step guide to phylogeny reconstruction* (cit. a p. 5).

[6] *Rosalind - Glossary*. URL: `http://rosalind.info/glossary` (cit. a p. 5).

[7] Christian Blum e Paola Festa. *Metaheuristics for String Problems in Bioinformatics* (cit. alle pp. 6, 26).

[8] David Sankoff. *Genome rearrangement with gene families* (cit. a p. 7).

[9] P. Bonizzoni et al. *Exemplar Longest Common Subsequence* (cit. a p. 8).

[10] Yin-Te Tsai. *The constrained longest common subsequence problem* (cit. a p. 8).

[11] P. Bonizzoni et al. *Variants of Constrained Longest Common Subsequence* (cit. a p. 8).

[12] *Complexity Zoo*. URL: `https://complexityzoo.uwaterloo.ca/Complexity_Zoo` (cit. a p. 11).

[13] C. H. Papadimitriou e M. Yannakakis. *Optimization, approximation, and complexity classes* (cit. a p. 11).

[14] G. Ausiello et al. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties* (cit. a p. 11).

[15] R. S. Mincu e A. Popa. *Better heuristic algorithms for the Repetition Free LCS and other variants* (cit. alle pp. 15, 16, 21, 22, 23).

[16] Paola Festa. *Ottimizzazione Combinatoria* (cit. a p. 16).

[17] C. G. Fernandes e M. Kiwi. *Repetition-free longest common subsequence of random sequences* (cit. alle pp. 16, 70).

[18] C. E. Ferreira e C. Tjandraatmadja. *A branch-and-cut approach to the repetition-free longest common subsequence problem* (cit. alle pp. 24, 50).

[19] M. Castelli, S. Beretta e L. Vanneschi. *A hybrid genetic algorithm for the repetition free longest common subsequence problem* (cit. alle pp. 26, 53).

[20] Christian Blum e Maria J. Blesa. *A comprehensive comparison of metaheuristics for the repetition-free longest common subsequence problem* (cit. alle pp. 26, 48, 53).

[21] T. A. Feo e M. G. C. Resende. *Greedy Randomized Adaptive Search Procedures* (cit. a p. 31).

[22] D. Ferone et al. *Enhancing and extending the classical GRASP framework with biased randomisation and simulation* (cit. a p. 32).

[23] A. Grasas et al. *Biased randomization of heuristics using skewed probability distributions: A survey and some applications* (cit. alle pp. 32, 77).

[24] N. Nepomuceno, P. Pinheiro e A. L. V. Coelho. *Tackling the Container Loading Problem: A Hybrid Approach Based on Integer Linear Programming and Genetic Algorithms* (cit. a p. 36).

[25] C. Blum et al. *Hybrid metaheuristics in combinatorial optimization: A survey* (cit. a p. 36).

[26] R. Martí, P. M. Pardalos e M. G. C. Resende. *Handbook of Heuristics* (cit. a p. 40).

[27] M. Birattari et al. *F-race and iterated F-race: An overview* (cit. a p. 52).

[28] M. López-Ibáñez et al. *The irace package: Iterated Racing for Automatic Algorithm Configuration* (cit. a p. 52).

[29] J. W. Hunt e T. G. Szymanski. *A Fast Algorithm for Computing Longest Common Subsequences* (cit. a p. 74).

[30] G. Yona et al. *Effective similarity measures for expression profiles* (cit. a p. 75).