UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA
DOCTORADO EN CIENCIAS DE LA COMPUTACIÓN

ALGORITMOS PARA EL CÁLCULO DE LA CADENA MEDIA

Profesores Guía: **Diego Seco Naveiras**
Departamento de Ingeniería Informática
y Ciencias de la Computación
Facultad de Ingeniería
Universidad de Concepción,
José Ignacio Abreu Salas
Departamento de Ingeniería Informática
Facultad de Ingeniería
Universidad Católica de la Santísima Concepción

Tesis para ser presentada a la Dirección de Postgrado de la Universidad de Concepción

PEDRO ANIEL SÁNCHEZ MIRABAL

CONCEPCIÓN - CHILE Noviembre 2019

A mi hijo Liam...
A mi esposa Greyyaremy...
A mi madre Marlene...
A mi padre Pedro Jesús...

Agradecimientos

Esta tesis no hubiera sido posible sin la ayuda de una gran cantidad de personas y entidades. Siendo consciente de que será muy difícil mencionar a todas las personas que de un modo u otro contribuyeron a este logro, parto agradeciéndoles a todos los familiares, amigos y colegas que, aunque no estén nombrados explícitamente, contribuyeron tanto con conocimientos como con palabras de ánimo o consejos en los momentos más oportunos. Les agradezco especialmente a mis tutores, Diego Seco y José Abreu por todo su tiempo y dedicación a lo largo de estos años, han contribuido significativamente en mi crecimiento profesional y personal. A todos los profesores del Departamento de Ingeniería Informática y Ciencias de la Computación de la UDEC, de forma muy especial a los que me apoyaron en los primeros momentos, incluso antes de mi llegada a Chile, como Loreto Bravo y Andrea Rodríguez, también a los que desde el principio me acogieron haciéndome sentir uno más del grupo, Leo Ferres, Gonzalo Rojas y Roberto Asín, con los que la relación trascendía lo estrictamente profesional, gracias por su amistad. Al profesor Yussef Farrán, que me apoyó en mis inicios como docente en Chile. A la profesora Lilian Salinas por su paciencia y el ánimo brindado en los últimos semestres. Al resto de los profesores que participaron en mi proceso formativo, estando presente en seminarios, defensas de proyectos, exámenes de califica-

ción y la propuesta de tesis, especialmente a Cecilia Hernández por su aporte. No puedo dejar de mencionar al personal administrativo del departamento, siempre cercanos, Hugo, Johanna y Carolina. A todos los estudiantes de nuestro programa de doctorado, dispuestos siempre a debatir ideas y aportar otras vías de solución, a los primeros graduados Diego Caro, José Fuentes y Erick Elejalde, y los que les deseo que se gradúen sin contratiempos, Isyed Rodríguez, Luis Cabrera, Narciso López, José Luis Viloria y Mabel Vidal. A todos con los que compartí mis pasantías por la Universidad de la Coruña, Susana, Óscar, Nieves y a las charlas y paseos con los del grupo de Comidas Enxenio, a Alex, Tirso, Cristina, Adrián, Carlos, en fin, a todos.

A los familiares y amigos, los de verdad, que apoyaron siempre en los momentos decisivos. Los que estuvieron ahí, cuando hizo falta tirar un cable a tierra para no volverse loco.

Finalmente, agradezco a la Comisión Nacional de Investigación Científica y Tecnológica (CONICYT), que me apoyó con la Beca Doctorado Nacional para Extranjeros 2014 Folio: 63140074. También al proyecto BIRDS (H2020 EU PROJECT MSCA GA NO 690941).

Resumen

El problema de la cadena media es NP-hard para muchas de sus formulaciones, específicamente cuando se emplea la distancia de edición de Levenshtein. Las heurísticas más competitivas aplicadas a este problema usan algoritmos basados en hacer perturbaciones sucesivas a una solución inicial, estos algoritmos son llamados *perturbation-based iterative algorithms*. La cadena inicial y la política para establecer un orden entre las posibles ediciones son esenciales en la eficiencia de estos algoritmos. En esta tesis se abordan estos sub-problemas. Primero, se estudia cómo una mejor clasificación de las posibles perturbaciones atendiendo a su calidad, tiene efectos en la convergencia de los algoritmos, como consecuencia de esto, se obtiene un nuevo ranking de perturbaciones, que no solamente tiene en cuenta lo que ocurre en relación con las cadenas en las que dicha perturbación está presente, si no, también con respecto al resto de las cadenas del conjunto. Luego, se muestra como la cadena inicial influye en el proceso, analizando principalmente dos variantes de inicialización para estos algoritmos, la cadena perteneciente al conjunto con menos distancia al resto, y la media de un subconjunto de la entrada del algoritmo. Este subconjunto mencionado anteriormente es determinado por los vecinos obtenidos al aplicar el algoritmo *Half Space Proximal* (HSP) a la cadena mediana del conjunto. De la misma forma, se proponen alternativas para

la reducción del conjunto de datos a procesar, sustituyéndolo por un subconjunto de éste, con el fin de disminuir los costes computacionales de los algoritmos. Finalmente, se valida la calidad de los algoritmos obtenidos mediante algoritmos de optimización que resuelven este problema de forma óptima. En el proceso de validación se lleva a cabo un gran número de experimentos que dan soporte a las conclusiones obtenidas. En estos experimentos se muestra que el error de los algoritmos diseñados se encuentra bien acotado, obteniendo soluciones óptimas para los casos en que SAT o ILP logran determinar el óptimo. En los casos en que el óptimo no es determinado por SAT o ILP, la solución encontrada es mejor o igual, logrando obtenerla en un tiempo de cómputo menor.

Índice general

Índice de figuras

Nota: Todas las figuras en este documento fueron creadas por el autor.

Índice de tablas

Nota: Todas las tablas en este documento fueron creadas por el autor.

Capítulo 1

Introducción y Motivación

1.1. Introducción

Son muchos los dominios donde las cadenas de caracteres son utilizadas, por citar solo algunos ejemplos relevantes, se pueden mencionar campos como la biología molecular o el reconocimiento de patrones. En muchos de los casos, las cadenas de caracteres son utilizadas para codificar instancias. En el ámbito biológico, es natural codificar mediante cadenas de caracteres las secuencias de ADN [1], de ARN [2] o los aminoácidos que conforman las proteínas [3]. En el ámbito del reconocimiento de patrones, entre otras alternativas, se pueden utilizar para la representación de contornos utilizando cadenas de Freeman de ocho direcciones [4].

Cuando se trabaja en problemas donde la información está representada a partir de cadenas de caracteres, es común la necesidad de encontrar la media de un conjunto de cadenas. Como fue definido en [5], la cadena media de un conjunto S de cadenas, es una cadena, no necesariamente perteneciente a S, que minimiza la suma de las distancias a cada uno de los elementos del conjunto para alguna definición de distancia. La cadena media es utilizada con fre-

cuencia en diversos contextos como representante de un conjunto de objetos.

La cadena media tiene aplicaciones en muchos dominios. Se ha utilizado en el reconocimiento de caracteres ópticos [6], en ese caso se generaliza el concepto de la media de grafos, donde se ven los caracteres que conforman una cadena como nodos de los mismos. En el trabajo citado se utilizan algoritmos genéticos para encontrar la media de un conjunto de cadenas, la que es utilizada como prototipos de contornos.

Otro uso que pudiera tener la cadena media es en trabajos donde se representan contornos de formas en dos dimensiones, similares a [7, 8], ya que en esos trabajos es necesario aplicar técnicas de agrupamiento [9], donde es necesario calcular la distancia a cada uno de los clusters a partir de la determinación de su centroide, como es el caso del algoritmo *k-means* [10].

De manera similar, la cadena media tiene aplicaciones cuando se emplean algoritmos de clasificación sobre conjuntos de datos desbalanceados, pues en estos casos, se intenta rebalancear las clases menos representadas en esos conjuntos de datos a partir de la incorporación de instancias construidas artificialmente [11]. También, siguiendo esta misma idea, se trata de disminuir la presencia de las clases más representadas, obteniendo a partir de la media de varias instancias una instancia nueva, que reemplaza en el conjunto de datos a las instancias a partir de las cuales se construyó [12].

Incluso, ha tenido aplicación en el reconocimiento automático de voz mediante los mapas auto-organizados de cadenas [13, 14, 15], o en la variante online de ese mismo problema, como es el caso de [16].

En la biología molecular la cadena media se emplea en el alineamiento de secuencias y la búsqueda de homólogos [17], donde

los avances en las técnicas de secuenciación han generado grandes volúmenes de datos que necesitan ser procesados [18, 19].

Otro campo que ha recibido una atención creciente es el de la compresión relativa [20, 21], en la que la compresión es lograda mediante el análisis de las subsecuencias que se comparten con una secuencia de referencia, donde se podría utilizar cadena media. Según se señala en [22, 23, 24], los algoritmos de compresión relativa son rápidos y logran grados altos de compresión, sin embargo, escoger una secuencia de referencia adecuada es un problema clave.

Debido a que para poder obtener la cadena media es necesario previamente tener establecido bajo qué tipo de distancia se está planteando el problema, es conveniente mencionar las principales distancias que se utilizan para el trabajo con cadenas de caracteres. Se han propuesto varias funciones para extender el concepto de distancia, métricas como la distancia de Hamming [25], la distancia de Levenshtein [26] y la distancia de Damerau-Levenshtein [27] son capaces de caracterizar la distancia entre cadenas en función de las operaciones de edición que pueden aplicarse sobre una cadena para convertirla en otra. La distancia de Hamming es aplicada a cadenas de igual longitud, donde se tiene en cuenta cuántos caracteres que ocupan la misma posición difieren en las dos cadenas. La distancia de Hamming ha sido utilizada desde los años 50 para la detección y la corrección de errores en códigos [25]. La distancia de Levenshtein [26, 28], está dirigida a transformar, a partir de operaciones de edición, una cadena S_1 en una cadena S_2. Las operaciones de edición contempladas por esta distancia son las inserciones, los borrados y las sustituciones, donde cada uno de los mismos tiene costo 1. En el caso más general de esta distancia, las diferentes operaciones de edición pueden tener un costo diferente [29, 30]. Esto es posible si se utiliza un esquema de ponderación

diferente o si se utiliza una matriz de pesos, que sea capaz de describir el costo de cada operación. La lista de operaciones de edición capaz de transformar una cadena en otra es conocida como transcripción de edición [29], la cual no es necesariamente única. El costo E_Q de una de estas transcripciones $Q = \{q_1, q_2, ..., q_n\}$ es calculado a partir de la suma de los costos individuales q_i. La distancia de edición entre dos cadenas bajo la distancia de edición de Levenshtein es definido como $D(S_1, S_2) = argmin_Q\{E_Q\}$. Extendiendo la formalización de la distancia de edición de Levenshtein, se encuentra la distancia de edición de Damerau-Levenshtein [27], en la cual, además de las operaciones de inserción, borrado y sustitución se incorpora la posibilidad de intercambio entre los caracteres de distintas posiciones de las cadenas, con un esquema que puede penalizar también la separación entre las secuencias que divergen. Además, puede tener control sobre la cantidad máxima de operaciones de edición de un mismo tipo aplicadas consecutivamente, dígase inserciones o borrados [29], pudiendo expresar su definición como $D(S_1, S_2) = argmin_Q\{E_Q - W_g(\#Q)\}$ donde $\#Q$ represente las separaciones Q y $W_g(\#Q)$ es una función de penalización.

Esta tesis se centra en el caso de la cadena media bajo la distancia de edición de Levenshtein, por ser la más estudiada en este contexto. Dentro de las primeras aproximaciones a este problema, quedó demostrado que el costo de encontrar una solución para la media de N cadenas de longitud l es $\mathcal{O}(l^N)$ [17], una complejidad computacional impráctica para la mayoría de las aplicaciones de este problema.

El cálculo de la cadena media para distancia de Levenshtein es un problema NP-Completo para muchas de sus formulaciones, como se muestra en [31, 32, 33]. En el caso de [34], se demuestra para alfabetos no acotados, entendiendo por no acotados alfabetos que

no tienen un tamaño fijo, a los que se les puede incorporar símbolos que no se conocen a priori. En [32] esto se demuestra para alfabetos de tamaño 7, y luego [33] lo generaliza para cualquier alfabeto finito, esta demostración es hecha mediante la reducción del problema de la cadena media al problema de la mayor subsecuencia común *(Longest Common Subsequence)*, problema para el que está demostrado que es NP-Completo [35].

El hecho de que se trate de un problema NP-Completo ha causado que haya sido enfrentado desde distintos enfoques. Uno de ellos consiste en la proyección de las cadenas a un espacio vectorial, donde se realiza el cálculo de la media para luego proyectar de vuelta el vector resultante aproximándolo a la cadena media [36, 37]. También han sido empleados modelos que utilizan programación lineal, donde se va acotando la distancia a la cadena media a partir de la distancia entre las cadenas del conjunto y el cumplimiento de la desigualdad triangular, dentro de este enfoque destacan [38, 39, 40]. De igual modo, existen trabajos donde este problema se trata bajo la mirada de los algoritmos evolutivos, como es el caso de [6] o el refinamiento iterativo basado en perturbaciones [5, 41, 14, 42, 43, 44, 45].

En el refinamiento iterativo basado en perturbaciones, se hacen modificaciones a una cadena inicial, con cada modificación se va mejorando la solución parcial, este proceso se repite hasta llegar a un punto en que ninguna perturbación ocasione mejoras. Una parte fundamental de este enfoque consiste en seleccionar la cadena inicial. Los mejores algoritmos existentes usan como cadena inicial la cadena vacía o la cadena perteneciente al conjunto que menos distancia acumule al resto de cadenas. Hasta ahora, los mejores resultados para este enfoque se han obtenido partiendo de la media del conjunto [45], entendida como la cadena perteneciente al conjunto

que acumula menor distancia al resto de cadenas bajo las condiciones de una distancia de edición determinada. En esta tesis, además de tomar como punto inicial los antes mencionados, se propone un nuevo punto de partida, calculado a partir de algunos vecinos de la media perteneciente al conjunto. Otro aspecto sobre el que esta tesis quiere arrojar luz, es en la obtención de una mejor evaluación del efecto de aplicar una posible perturbación, retomando la idea expuesta en [14, 43]. Esto es fundamental, ya que en aras de ahorrar tiempo de cómputo, probar ciertas perturbaciones con muy escasa probabilidad de éxito, no amerita el esfuerzo computacional. Encontrar un nuevo método para estimar la calidad de una posible operación podría logra una reducción de la cantidad de distancias de edición que es necesario calcular. También, resultaría interesante evaluar variantes de reducción del problema, centrándolo en un subconjunto del total de cadenas, específicamente en aquellas que sean lo suficientemente representativas del resto, estudiando cómo establecer un balance correcto entre la cantidad de cadenas utilizadas, el tiempo de cómputo y la calidad de la solución que se obtiene.

1.2. Hipótesis

El problema de encontrar la cadena media de un conjunto de cadenas, bajo las condiciones de la distancia de edición de Levenshtein, ha sido abordado desde distintos enfoques. Sin embargo, los algoritmos que obtienen una mejor aproximación a la cadena media todavía conllevan un gran número de cálculos para encontrarla. Las hipótesis en las que se basa este trabajo de tesis son:

- H1) Estimar el efecto de las posibles ediciones en la búsqueda de la cadena media conllevaría una disminución en la cantidad de distancias de edición calculadas conservando la calidad de

la media. El tiempo de obtener dicha estimación no puede ser mayor que el tiempo de cálculo de las distancias de edición que pretende disminuir.

- H2) Es posible reducir la cantidad de distancias de edición calculadas si se centra la búsqueda en un subconjunto del conjunto inicial que sea capaz de representarlo adecuadamente, ponderando aquellos elementos similares que estén sobrerrepresentados.

- H3) El punto de partida del algoritmo tiene efectos en el esfuerzo computacional para obtener un resultado, pudiendo existir alternativas como punto de partida que sean mejores que la cadena perteneciente al conjunto que menos distancia acumule al resto.

1.3. Objetivo General

El objetivo general de esta tesis es:

Desarrollar nuevos algoritmos que mejoren el estado del arte para las aproximaciones heurísticas para el cálculo de la cadena media bajo las condiciones de la distancia de edición de Levenshtein en términos de velocidad de convergencia, manteniendo la calidad de la solución.

1.4. Objetivos Específicos

En aras del cumplimiento exitoso del objetivo general, se proponen los siguientes objetivos específicos:

G1: Estimar el efecto de una perturbación en la minimización de la expresión que define la cadena media. Con este objetivo se

pretende obtener una mayor información a priori del resultado de aplicar una perturbación a la cadena candidata, de forma tal que se pueda centrar el esfuerzo computacional en aquellas perturbaciones más prometedoras.

G2: Aplicar el enfoque de Selección de Pivotes sobre Espacios Métricos al problema de la cadena media. Este objetivo está encaminado a reducir el espacio de búsqueda, como una forma de aligerar la carga computacional que implica trabajar con el total de las cadenas del conjunto. La clave estará en poderlo hacer sin perder calidad en la solución a la que se llegue.

G3: Explorar alternativas distintas a la cadena vacía y a la cadena del conjunto que menos distancia acumule al resto como punto inicial de los algoritmos. El punto de partida del algoritmo influye en la calidad de la solución y el tiempo de ejecución de los algoritmos, hasta ahora no hay ninguna demostración formal de que el mejor punto de partida sea la cadena del conjunto que menos distancia acumule al resto, por lo que se hace muy interesante intentar otras variantes de inicialización para estos algoritmos.

G4: Validar la cota de error de los algoritmos diseñados contra otros enfoques que para instancias pequeñas encuentren la solución óptima. Para este fin se utilizarán técnicas de optimización como la satisfacibilidad booleana y la Programación Lineal Entera.

1.5. Metodología

Para cumplir los objetivos específicos anteriormente expuestos se propuso el conjunto de tareas que se exponen a continuación:

T1: Selección de bases de datos de dominio público para el desarrollo de los experimentos para el cálculo de la cadena media. Se propone la utilización tanto de datos sintéticos como reales. Además, las bases de datos deben ser heterogéneas, representando dominios con características diferentes en cuanto a alfabeto, longitud de las cadenas y número de cadenas.

T2: Diseño e implementación de nuevos algoritmos para el problema de la cadena media en base a las hipótesis planteadas en la Sección 1.2.

T3: Evaluación comparativa de los algoritmos desarrollados, comparándolos con el estado del arte en la materia, teniendo en cuenta la velocidad de convergencia y la calidad de la solución encontrada. La velocidad de convergencia se expresará en términos de número de operaciones y tiempo de cómputo, mientras que la calidad de la solución se evaluará a partir del porcentaje de reducción de la distancia total entre el punto de partida del algoritmo y el punto de parada.

T4: Modificación de los algoritmos obtenidos en la tarea 2 para incluir el concepto de pesos en las distintas cadenas.

T5: Diseño e implementación de nuevos algoritmos para el problema de la cadena media utilizando específicamente el enfoque de selección de pivotes en espacios métricos.

T6: Evaluación comparativa de los algoritmos de la tarea 2 con los de la tarea 5, teniendo en cuenta la velocidad de convergencia y la calidad de la solución encontrada.

T7: Exploración de puntos de partida alternativos para los algoritmos de cadena media.

T8: Reducción de las alternativas de edición mediante la poda de la lista de operaciones a partir de su valor de calidad esperado.

T9: Diseño e implementación de nuevos algoritmos para el problema de la cadena media utilizando los resultados obtenidos en las tareas 7 y 8.

T10: Validación de la cota de error de los algoritmos diseñados contra otros enfoques que para instancias pequeñas encuentren la solución óptima utilizando satisfacibilidad booleana *(SAT solvers)*.

T11: Validación la cota de error de los algoritmos diseñados contra otros enfoques que para instancias pequeñas encuentren la solución óptima utilizando algoritmos de Programación Lineal Entera *(ILP)*.

T12: Establecimiento de un marco comparativo, mediante un sistema en línea, en el que se brinde la posibilidad de comparar nuevos algoritmos para el cálculo de la cadena media con variantes desarrolladas en esta tesis y otros enfoque también mencionados.

T13: Compilación de todos los resultados, análisis, conclusiones y discusión de las mismas.

1.6. Estructura de la tesis

A continuación se hará una descripción de cómo está estructurado el documento a partir de este punto. Es importante aclarar que la modalidad de esta tesis es "por compendio de publicaciones"[1]. Por tanto, se prosigue con el Capítulo 2, donde se describen las distintas publicaciones que ha generado nuestra investigación, partiendo por la Sección. 2.1 donde se expone un artículo que fue publicado con fecha 1 de abril de 2019 en la revista Pattern Recognition Letters, Volumen 120, páginas 107-114, siendo sus autores P. Mirabal, autor de esta tesis, y sus tutores J. Abreu y D. Seco. Este artículo tributa directamente a la validación de nuestra hipótesis número uno: "Estimar el efecto de las posibles ediciones en la búsqueda de la cadena media traería aparejado una disminución en la cantidad de distancias de edición calculadas conservando la calidad de la media. El tiempo de obtener dicha estimación no puede ser mayor que el tiempo de cálculo de las distancias de edición que pretende disminuir.", al objetivo específico número 1: "Estimar el efecto de una perturbación en la minimización de la expresión que define la cadena mediaz a las Tareas 1, 2 y 3 descritas en la Metodología. El principal aporte de este artículo, es que en él se logra mejorar el estado del arte para este tipo de algoritmos en términos de velocidad de convergencia al proponer una forma más efectiva para medir el efecto de una perturbación en la minimización de la expresión que define la cadena media.

Luego, en la Sección. 2.2 se presenta un artículo presentado en las Jornadas Chilenas de Computación 2019, las memorias de este congreso serán publicadas por IEEE e indexadas en Scopus. Sus au-

[1]Según lo establecido en el Artículo 13 del Reglamento del Doctorado en Ciencias de la Computación de la Universidad de Concepción http://www.inf.udec.cl/postgrado-2/documentos-y-formularios/

tores son P. Mirabal, autor de esta tesis, junto con J. Abreu y O. Pedreira. Este artículo tributa directamente a la validación de nuestra hipótesis número dos: "Es posible reducir la cantidad de distancias de edición calculadas si se centra la búsqueda en un subconjunto del conjunto inicial que sea capaz de representarlo adecuadamente, ponderando aquellos elementos similares que estén sobrerrepresentados.", al objetivo específico número 2: "Aplicar el enfoque de Selección de Pivotes sobre Espacios Métricos al problema de la cadena media.$_\mathcal{Z}$ a las Tareas 4, 5 y 6 descritas en la Metodología. El principal aporte de este artículo, es que en él se logra una reducción de un 8 % como promedio en la cantidad de operaciones respecto al artículo presentado en la Sección. 2.1. Entre las principales modificaciones hechas al nuevo algoritmo, se incluye la posibilidad de ponderar las distintas cadenas, característica que no estaba presente en los algoritmos anteriores.

Continuando con la Sección 2.3, donde se presenta un artículo que está en fase de revisión en la revista Pattern Recognition Letters, siendo sus autores P. Mirabal, autor de esta tesis, y sus tutores J. Abreu y D. Seco, junto con O. Pedreira y E. Chávez[2]. Este artículo tributa directamente a la validación de nuestra hipótesis número tres: "El punto de partida del algoritmo tiene efectos en el esfuerzo computacional para obtener un resultado. Existen alternativas mejores que la media del conjunto.", al objetivo específico número 3: "Explorar alternativas distintas a la cadena vacía y a la cadena del conjunto que menos distancia acumule al resto como punto inicial de los algoritmos.$_\mathcal{Z}$ a las Tareas 7, 8 y 9 descritas en la

[2]La idea de explorar el uso de la Selección de Pivotes sobre Espacios Métricos se produjo a partir de una estancia de investigación en la Universidad de la Coruña, entre septiembre y diciembre de 2016. Esta estancia fue financiada por el programa de innovación e investigación de la Unión Europea Horizonte 2020, bajo el convenio Marie Skłodowska-Curie 690941.

Metodología. Hay dos aportes fundamentales que destacar de este artículo, el primero es la detección de un nuevo punto de partida para los algoritmos del cálculo de la cadena media. Este punto se obtiene mediante la aplicación del test *Half Space Proximal*[46] a la cadena perteneciente al conjunto que menor cantidad de distancia acumula al resto. El segundo, es que se ha podido establecer a partir de la valoración de las posibles ediciones, puntos a partir de los cuales se pueda podar la lista de candidatos reduciendo así el espacio de búsqueda, lo que conlleva a que los algoritmos converjan más rápidamente.

Seguidamente, en la Sección. 2.4, se expone el artículo titulado Median String Challenge, que se encuentra en fase de revisión en la revista *Expert Systems with Applications*. Este artículo tiene como autores a R. Asin, profesor de la Universidad de Concepción, a los tutores de este trabajo, J. Abreu y D. Seco, junto con su autor, P. Mirabal. En él, se presenta una plataforma informática, que brinda la posibilidad de reportar los resultados de distintos algoritmos para el cálculo de la cadena media, este reto consiste en diversas variantes de este problema. Resulta interesante para esta tesis la comparación de los algoritmos con el enfoque basado en perturbaciones, *perturbation-based iterative algorithms*, frente a los que utilizan programación lineal, ya que los casos donde estos últimos encuentran la solución óptima, permiten validar la calidad de la solución encontrada por los primeros. Este artículo tributa al objetivo específico número 4: "Validar la cota de error de los algoritmos diseñados contra otros enfoques que para instancias pequeñas encuentren la solución óptima. Para este fin se utilizarán técnicas de optimización como la satisfacibilidad booleana y la Programación Lineal Entera.z a las Tareas 10, 11 y 12 descritas en la Metodología.

Por último, en el Capítulo 3, a partir de los resultados obtenidos,

se hace una discusión de estos, se exponen las conclusiones a las que se ha llegado, esto tributa a la Tarea 13, y también se hace un recuento de aquello que aunque importante, no se ha podido abordar en este trabajo quedando como trabajo futuro.

Capítulo 2

Producción Científica

En este capítulo se expondrán cada uno de los trabajos que han sido enviados a revistas o congresos como fruto de esta investigación, en cada uno de los casos se expondrá los detalles de la revista o congreso en cuestión, así como un resumen del contenido tratado en el artículo.

2.1. Assessing the best edit.

Título: *Assessing the best edit in perturbation-based iterative refinement algorithms to compute the median string.*

2.1.1. Abstract.

Different pattern recognition techniques such as clustering, k-Nearest Neighbors classification or instance reduction algorithms require prototypes to represent pattern classes. In many applications, strings are used to encode instances, for example, in contour representations or in biological data such as DNA, RNA and protein sequences. Median strings have been used as representatives

of a set of strings in different domains. Finding the median string is an NP-Complete problem for several formulations. Alternatively, heuristic approaches that iteratively refine an initial coarse solution by applying edit operations have been proposed. We propose here a novel algorithm that outperforms state of the art heuristic approximations to the median string in terms of convergence speed by estimating the effect of a perturbation in the minimization of the expressions that define the median strings. We present comparative experiments to validate these results.

2.1.2. Introduction

The concept of *median* is useful in many contexts as a representative for a collection of objects. As defined in [5] the *median* of a set S of strings is the one that minimizes the sum of the distances to each element in the collection. In the case of strings, Levenshtein distance [26] has been widely used.

For edit distance metrics, [31] and [33] show that computing the median of a set of strings is a problem within the NP-Complete class for several formulations. Different approximations have therefore been proposed. One of those approaches, called perturbation-based iterative refinement by [41], has been studied in [5], [14], [42], and [43]. The kernel idea is to perform successive edit operations to an initial string while at least one of the perturbations leads to an improvement. Those approaches have proved to converge to quality approximations of the true median by [42], yet they may require performing an important number of perturbations before converging. For these methods, it is important to study how to score the goodness of each candidate perturbation in order to try the most promissory ones.

2.1.3. Preliminaries

Let Σ be an alphabet where ϵ denotes the empty symbol, Σ^* the set of all strings over Σ, also S_i, $S_j \in \Sigma^*$. An edit operation is a pair $(a, b) \neq (\epsilon, \epsilon)$, written $a \to b$, which transforms a string S_i into S_j, if $S_i = \sigma a \tau$ and $S_j = \sigma b \tau$, where σ and τ represent substrings. Substitutions, deletions and insertions are denoted as $a \to b$, $a \to \epsilon$, and $\epsilon \to b$, respectively. Let $E_{S_i}^{S_j} = \{e_1, e_2, ..., e_n\}$ be a sequence of edit operations transforming S_i into S_j and, $\omega(a \to b)$ a domain-specific function that assigns a cost to an edit operation. The cost of E is $\omega(E) = \sum_{e_i \in E} \omega(e_i)$, and the edit distance from S_i to S_j, is defined as $d(S_i, S_j) = argmin_E\{\omega(E_{S_i}^{S_j})\}$.

The *median string* is the one that minimizes $\sum_{S_i \in S} d(\hat{S}, S_i) | \hat{S} \in \Sigma^*$. A common approximation to the true median string is the *set median*, a string that minimizes $\sum_{S_i \in S} d(\hat{S}, S_i) | \hat{S} \in S$. It is not necessary for neither the median string nor the set median to be unique.

2.1.4. Related work

Results in a seminal work, [17], allow the computation of the median of a set S of N strings in $\mathcal{O}(l^N)$ for the Levenshtein metric and strings of length l. However, the computational burden required makes this approach impractical in most scenarios. Therefore, different heuristics have been proposed to overcome this issue with the aim of reducing the search space.

A general strategy is to build the approximate median, one symbol at a time, from an empty string. A goodness function must be defined in order to decide which symbol is the next to be appended. A greedy implementation is described in [31]. In [47] a tie-breaking criterion is presented when many symbols have the same goodness

index.

An alternative approach is to build the approximate median by successive refinements of an initial string such as the set median, or the greedy approximation in [31].

Starting from the set median, [5] systematically changes the current solution performing insertions, deletions and substitutions in every position. An edit is accepted if it leads to an improvement. One specific order to apply operations is proposed in [42]. Let $\hat{S}^{t-1}$ be the approximated median at step $t-1$, two possible alternatives are considered in the application of edit operations. The first one, performs substitutions in each position of $\hat{S}^{t-1}$, testing each possible symbol. The new partial solution $\hat{S}^t$ is the string, selected from all the new candidates that minimizes $\sum_{S_i \in S} d(\hat{S}^t, S_i)$. A similar procedure is repeated for deletions, substitutions, and insertions, in such order and until there are no more improvements.

Another variation is to generate a number of candidate approximations from $\hat{S}^{t-1}$ by separately applying substitutions, deletions, and insertions at position i. The candidate will be the string with lowest $\sum_{S_i \in S} d(\hat{S}^t, S_i)$, if it is also lower than $\sum_{S_i \in S} d(\hat{S}^{t-1}, S_i)$. If no such a sequence exists, $\hat{S}^{t-1}$ remains as the best approximation found so far. In the next step, the algorithm repeats the search, but from position $i+1$. The search stops when there are no more improvements. Theoretical and empirical results demonstrate that this approach achieves good approximations to the true median string.

It is important to note that, to evaluate the goodness of an operation, the approaches discussed above require to compute the distance from the candidate median to the strings in the set. No a priori ranking of operations is provided. A core problem is to determine the best operation but with a computational cost lower

than the one required to compute all the distances from the new candidate to each string.

An alternative to speed up the computation of the approximated median string is described in [48]. Some operations are preferred, for example, not all possible substitutions are evaluated. Looking at the cost of substitutions, the two closest symbols to the examined position are selected, and only those symbols are tested as viable substitutions.

Instead of applying operations one by one, [14] and [43] propose to apply multiple perturbations at once. Results in [44] suggest that this approach has a faster convergence but the quality of the approximated median is worse.

[44] take advantage of results in [6] to study how to score the goodness of each candidate perturbation in order to test first the most promissory ones. Let $\hat{S}^{t-1}$ be the approximated median at step $t-1$, e_k an edit operation and $\hat{S}^t$ the string derived from $\hat{S}^{t-1}$ by applying e_k. Possible edit operations are ranked by the number of strings for whom $e_k \in \hat{E^{S^{t-1}}_{Sk}}$. Results show that the proposed approach drastically improves the convergence speed while preserving the quality of the approximated median in comparison with [48]. However, whether $\sum_{S_i \in S} d(\hat{S}^t, S_i) \leq \sum_{S_i \in S} d(\hat{S}^{t-1}, S_i)$ or not is also determined by strings for whom $e_k \notin \hat{E^{S^{t-1}}_{Sk}}$, which are not explicitly considered by the authors.

The median string problem has also been studied from other perspectives quite different from those of our work. In [38] a Linear Programming model for the median string problem is proposed, providing a lower bound and analyzing the cases where the true median cannot be achieved. However, the procedure does not compute the median string. Recently, [39, 40] developed Integer Linear Programming approaches for the median and center string problems, in

which special cases were considered when the dissimilarity measure satisfies the triangle inequality and when the distance between the strings in S is bounded.

Some authors, such as [49], have studied the approximation to the median string not only under the Levenshtein edit distance but also under the stochastic edit distance [50]. In other works, the search for the approximate median is not performed directly in the string space but in a vector space in which the strings are embedded; this is the approach studied in [37], which also relies on the weighted median concept described by [6]. A different approach based on an evolutionary framework was proposed by [51].

Our main contribution, described in next section, is an improved heuristic to rank edit operations that shows similar or better results in terms of $\sum_{S_i \in S} d(\hat{S}, S_i)$, but demanding less computational effort.

2.1.5. A new algorithm for computing a quality approximate median string

Let $e_k \in E_{S_i}^{S_j}$ be the edit sequence from S_i to S_j with minimum cost, i.e. $d(S_i, S_j) = \omega(E)$, and S_i' the string derived from S_i by the application of e_k. Results in [6] allow to compute $d(S_i', S_j)$ as $d(S_i, S_j) - \omega(e_k)$. Thus, since $\omega(e_k) \geq 0$, then $d(S_i', S_j) \leq d(S_i, S_j)$.

For example, let $\hat{S}^{t-1}$ be the candidate median at step $t-1$ from strings in the set $S = \{S_1, S_2, S_3\}$ and $E_{S_1}^{S^{t-1}}$, $E_{S_2}^{S^{t-1}}$, $E_{S_3}^{S^{t-1}}$ the respective minimum cost edit sequences from $\hat{S}^{t-1}$ to each string in S. Suppose there is an edit operation such that $e_k \in E_{S_1}^{S^{t-1}} \cap E_{S_2}^{S^{t-1}}$, $e_k \notin E_{S_3}^{S^{t-1}}$ and $\hat{S}^t$ is derived from $\hat{S}^{t-1}$ by e_k. As regards of $\sum_{S_i \in S} d(\hat{S}^t, S_i) = d(\hat{S}^t, S_1) + d(\hat{S}^t, S_2) + d(\hat{S}^t, S_3)$, applying e_k could lead to a improvement since $d(\hat{S}^t, S_1) \leq d(\hat{S}^{t-1}, S_1)$, the same

holds for S_2. However, it is also determined by $d(\hat{S}^t, S_3)$.

In the example above, e_k splits S into two subsets, $S^{YES} = \{S_i \in S \mid d(\hat{S}^t, S_i) \leq d(\hat{S}^{t-1}, S_i)\}$ and $S^{NO} = S - S^{YES}$, thus $\sum_{S_i \in S} d(\hat{S}^t, S_i) = \sum_{S_i \in S^{YES}} d(\hat{S}^t, S_i) + \sum_{S_i \in S^{NO}} d(\hat{S}^t, S_i)$, and also $\sum_{S_i \in S^{YES}} d(\hat{S}^t, S_i) \leq \sum_{S_i \in S^{YES}} d(\hat{S}^{t-1}, S_i)$.

The heuristic proposed in [44] uses this information in order to rank the possible edit operations applicable to $\hat{S}^{t-1}$ to derive $\hat{S}^t$, selecting the one that could be expected to lead to the lowest $\sum_{S_i \in S} d(\hat{S}^t, S_i)$. However, it does not consider how $\sum_{S_i \in S^{NO}} d(\hat{S}^t, S_i)$ behaves. Also, authors only considered in S^{YES} strings S_j for whom $e_k \in E_{Sj}^{S^{t-1}}$, which guarantees that $d(\hat{S}^t, S_j) \leq d(\hat{S}^{t-1}, S_j)$. In this work we identify some strings S_i that authors assigned to S^{NO} for whom $d(\hat{S}^t, S_i) \leq d(\hat{S}^{t-1}, S_i)$ holds.

Heuristic to select the best edit operation

First, we discuss how to determine an upper bound for $d(\hat{S}^t, S_i)$, for some $S_i \in S^{NO}$ depending on the type of operation e_k and ω without computing the actual distance. We use this result to propose a new heuristic to select the edit operation that could be expected to minimize $\sum_{S_i \in S^{YES}} d(\hat{S}^{t-1}, S_i)$.

Lemma 1. *Let S_x, S_y and M be strings, with $S_x \in S$ and $S_y \in S$. Also, let $E_{S_x}^M$ and $E_{S_y}^M$ be the minimum cost edit sequences transforming M into S_x and S_y, respectively, $e_{iS_x}^M = (a \to b) \in E_{S_x}^M$ the operation to be applied to position i of M, $e_{iS_y}^M = (a \to c) \in E_{S_y}^M$ an edit operation of the same type as $e_{iS_x}^M$, and $\hat{M}$ derived from M by $e_{iS_x}^M$. If $\omega(b \to c) \leq \omega(a \to c)$, then $d(\hat{M}, S_y) \leq d(M, S_y)$.*

Demostración. Let $E_{S_y}^M = \{e'_1, e'_2, ..., (a \to c), ..., e'_m\}$. By definition of the edit distance, deriving $\hat{M}$ from M by $e_{iS_x}^M$ means that $M = \sigma a \tau$ and $\hat{M} = \sigma b \tau$. Thus, the edit sequence $E_{S_y}^{\hat{M}} = \{e'_1, e'_2, ..., (b \to c), ..., e'_m\}$

will transform $\hat{M}$ into S_y. In this case, we can write the cost of those edit sequences as equations (2.1) and (2.2) from which we get that, if $\omega(b \to c) \leq \omega(a \to c)$ then $\omega(E^{\hat{M}}_{S_y}) \leq \omega(E^{M}_{S_y})$.

$$\omega(E^{M}_{S_y}) = \sum_{i=0}^{l-1} \omega(e'_i) + \omega(a \to c) + \sum_{i=l+1}^{m} \omega(e'_i) \tag{2.1}$$

$$\omega(E^{\hat{M}}_{S_y}) = \sum_{i=0}^{l-1} \omega(e'_i) + \omega(b \to c) + \sum_{i=l+1}^{m} \omega(e'_i) \tag{2.2}$$

□

With the result in Lemma 1 we estimate the decrease Δ_i of $d(\hat{S}^t, S_i)$ after applying the operation $e_k = (a \to b)$. For each string in $\{S_i | e_k \in E^{\hat{S}^{t-1}}_{S_i}\}$ we set $\Delta_i = \omega(a \to b)$. Unlike [44] we also consider its *repercussion*[1] in each string in $\{S_i | e_k \notin E^{\hat{S}^{t-1}}_{S_i}\}$, in this case, we set $\Delta_i = (\omega(a \to c) - \omega(b \to c))$. Note that, $\Delta_i \geq 0$ for strings in which Lemma 1 holds. The best edit operation is the one that maximizes $\sum \Delta_i$.

An illustrative example

Let us consider an alphabet $\Sigma = \{0, 1, 2, 4, \epsilon\}$ and a cost function represented in a tabular form in Table 2.1:

Cuadro 2.1: Cost of Substitutions

	0	**1**	**2**	**4**	ϵ
0	-	1	2	4	2
1	1	-	1	3	2
2	2	1	-	2	2
4	4	3	2	-	2
ϵ	2	2	2	2	-

[1] Hence the name of our method.

Our current candidate string is $\hat{S}^{t-1}$ = "2". The strings in the set are S_1 = "0", S_2 = "1" and S_3 = "4", $\omega(E_{S_1}^{\hat{S}^{t-1}}) = \omega(2 \to 0) = 2$, $\omega(E_{S_2}^{\hat{S}^{t-1}}) = \omega(2 \to 1) = 1$ and $\omega(E_{S_3}^{\hat{S}^{t-1}}) = \omega(2 \to 4) = 2$. Now, $\sum_{S_i \in S} d(\hat{S}^{t-1}, S_i) = 5$, and we have to find $\sum_{S_i \in S} d(\hat{S}^{t}, S_i) < 5$. We can try three possible perturbations for $\hat{S}^t$: $(2 \to 0)$, $(2 \to 1)$ and $(2 \to 4)$. Table 2.2 shows how S is divided in S^{YES} and S^{NO} depending only on whether an operation is in the transformation sequence or not.

Cuadro 2.2: Pessimistic Analysis vs Repercussion Analysis

	Pessimistic Analysis		Repercussion Analysis	
e_k	S^{YES}	S^{NO}	S^{YES}	S^{NO}
e_1	S_1	S_2, S_3	S_1	S_2, S_3
e_2	S_2	S_1, S_3	S_1, S_2	S_3
e_3	S_3	S_1, S_2	S_3	S_1, S_2

In this case, $(2 \to 0)$ and $(2 \to 4)$ are tied, because both would directly decrease distance in 2, but the repercussion in the rest of S is not considered. However, taking a closer look at Table 2.1, we can see that if $(2 \to 1)$ is applied, both the distances to S_1 and to S_2 will decrease. Then, we want to check how one operation affects the rest. We summarize the analysis in Table 2.3. We can see in the last column that $(2 \to 1)$ is the best possible operation, reducing the total distance in 1.

Cuadro 2.3: Repercussion costs and new ranking. DE is the direct effect over $\sum_{S_i \in S} d(\hat{S}^{t-1}, S_i)$ and IE is the indirect effect.

e_k	DE	IE				
		$(2 \to 0)$	$(2 \to 1)$	$(2 \to 4)$	Rep.	Total
$(2 \to 0)$	2	0	0	-2	-2	0
$(2 \to 1)$	1	1	0	-1	0	1
$(2 \to 4)$	2	-2	-1	0	-3	-1

Computing the approximate median string

Algorithm 1 shows our method to compute the approximate median string.

Input : instance set S, initialization string R
Output : approximate median string $\hat{S}$

1 $R' = R$
2 **repeat**
3 $\quad \hat{S} = R'$
4 $\quad$ **foreach** $S_i \in S$ **do**
5 $\quad\quad$ compute $d(R', S_i)$ and the respective $E_{S_i}^{R'}$
6 $\quad\quad$ statistics.update($E_{S_i}^{R'}$)
7 $\quad$ **foreach** *position* j *of* R' **do**
8 $\quad\quad$ **foreach** *symbol* $si \in \Sigma$ **do**
9 $\quad\quad\quad$ **foreach** *symbol* $sj \in \Sigma$ **do**
10 $\quad\quad\quad\quad$ $Rep[j][si] += \omega(R[j] \rightarrow si) - \omega(si \rightarrow sj)$
11 $\quad$ Q: a queue of operations sorted by goodness index
12 $\quad$ **while** $\sum_{S_i \in S} d(\hat{S}, S_i) \leq \sum_{S_i \in S} d(R', S_i)$ **and** $Q \neq \emptyset$ **do**
13 $\quad\quad$ $e_k = Q.dequeue$
14 $\quad\quad$ obtain a new candidate R' applying e_k to $\hat{S}$
15 **until** *no operation* e_k *applied to* $\hat{S}$ *improve the result*
16 **return** $\hat{S}$

Algorithm 1: AppMedianStringRepercussion(S,R) :$\hat{S}$

This algorithm is based on the `AppMedianString` function presented in [44], being the main differences in lines 7-14. The original algorithm keeps the edit operations associated with each position of R' while computing the distance to each string in S, lines 4-6. All possible operations are stored in a priority queue Q sorted by goodness index, which is based on the statistics computed above. In lines 7-14, we collect the possible repercussion of each operation on the elements in S^{NO}, which is used to compute a better goodness index.

Computational cost analysis

In each iteration of the outer **repeat**, the algorithm computes the distance from R' to each $S_i \in S$, lines 4-6, which can be accomplished in $\mathcal{O}(N \times l^2)$ time using the algorithm in [52], where $N = |S|$ and l is the length of the longest string, which for simplicity we will assume is R'. Computing the *repercussion*, lines 7-14, for each symbol in R' requires testing all pairs of symbols in Σ. In the worst case, this is upper bounded by $O(l \times \Sigma^2)$.

Obtaining the sorted queue Q, line 15, requires time proportional to $\mathcal{O}(|Q| \log |Q|)$, an upper bound for $|Q|$ is given below.

For each position of R', $min\{N, |\sum|\}$ substitutions are possible, also at most l deletions can be applied in R'. For insertions, the worst case arises when R' is the empty string and only insertions are performed, which is upper bounded by $\sum_{S_i \in S} |S_i|$ or $N \times l$. Thus, a loose upper bound for $|Q|$ is $\mathcal{O}(N \times l)$.

The worst case for the last **while**, lines 16-19, occurs when all operations in Q are examined, computing the edit distance from R' to $S_i \in S$, which costs $N \times l^2$. Considering $|Q|$, this stage requires a time proportional to $\mathcal{O}(N \times l^3 \times (min\{N, |\sum|\}))$.

All stages are repeated until no edit operation leads to an improvement. Let k be the number of iterations of the outer **repeat**, thus the total time of the algorithm is bounded by $\mathcal{O}(k \times N \times l^3 \times (min\{N, |\sum|\}))$.

In the experimental evaluation, we show that this bound is rather pessimistic and our heuristic usually needs just a few operations per iteration. By providing a better ranking, we save on the number of operations explored per iteration, regardless that we expend some computations to bound the *repercussion*. This is a key difference with the algorithm by [44], which uses more operations per iteration.

2.1.6. Experimental results

We designed experiments to compare our proposal with respect to those in [44] for different alphabets, set sizes and string lengths. We are interested in studying the ratio: $\frac{\sum_{S_i \in S} d(\hat{S}, S_i)}{\sum_{S_i \in S} d(S^M, S_i)}$ where S^M is the set median. Also, we want to compare the number of edit operations, i.e. perturbations, that requires an algorithm to converge.

We used four different datasets. The first one, identified as NIST Freeman, corresponds to Freeman chain codes [53] which represents contours of letters from the *NIST-3* database, also used in [54, 55, 56, 44]. For the Freeman chain codes, the substitution cost between symbols is equivalent to one unit for every 45 degrees of difference in the orientation of each symbol, for insertions and deletions the cost is always of two units as in [57]. In this dataset, we evaluate sets of size $\{10, 45, 90, 180, 270, 360\}$, and for each one, 26 independent samples were drawn with average length ranging in [120..,264].

The second and third datasets, named Proteins and Insulin respectively, contain strings from an alphabet of 23 symbols representing amino acids. In both cases, we use the well-known BLOSUM62 cost function [58]. Proteins dataset contains strings with length ranging in [400..,600]. In this dataset, sets of size {45, 90, 180, 360, 720} were studied, but only one sample for each set size was selected. For the Insulin dataset, we selected samples of orthologs of insulin protein, representing 70 species, obtained from eggNog online aplication[2] with length ranging in [100..,300] and average 150. We evaluate sets of size$\{20, 40, 80, 120, 160\}$, in each case, we replicate the experiments 4 times. Finally, the fourth dataset, identified as Synthetic Freeman, contained randomly generated Freeman chain codes as in [44]. In this dataset, we aimed to study how algorithms

[2]http://eggnogdb.embl.de/#/app/home

scale with the average length of the strings. Average lengths of $\{20, 40, 80, 160, 320\}$ with a standard deviation of 10 % were considered. For each length, 4 independent samples of 20 strings were generated.

We compared the two approaches described in [44] with our proposal, labeled as *Frequency*, *Frequency*Cost* and *Repercussion*, respectively. All the experiments were performed in an Intel(R) Xeon(R) E5-2630 v3 @ 2.4GHz, 48GB RAM, running Ubuntu 16.04 (kernel 4.4.0-71). We compiled with Oracle JDK version 1.8.

As regards to the total number of edit operations explored, Fig. 2.1, Fig. 2.3, Fig. 2.5 and Fig. 2.7 show that the number of operations required by the algorithm to converge is significantly lower in our proposal for all datasets. Also, our heuristic scales better with the size of the dataset and the average length of the strings. It is worth to notice that each point in the series represents the total number of operations of all experiments for a specific set size or string length.

Despite analyzing the operations, we show how the execution time of these methods behaves. In Fig. 2.2, Fig. 2.4, Fig. 2.6 and Fig. 2.8, it can be seen that the actual computation time is proportional to the number of operations, shown in Fig. 2.1, Fig. 2.3 Fig. 2.7 and Fig. 2.5

As a sanity check, in Fig. 2.9, Fig. 2.11 and Fig. 2.12 we show that the quality of the median remains statistically equivalent in both Freeman chain codes datasets. However, in the Proteins dataset, we obtained a surprising result, as our proposal achieves a better median than the state of the art, as we can see in Fig. 2.10. This means that the other methods stall at a local optimum, except for the set of size 45. Also, this explains the graphic in Fig. 2.4 where the result for the dataset with 45 instances is consistent with

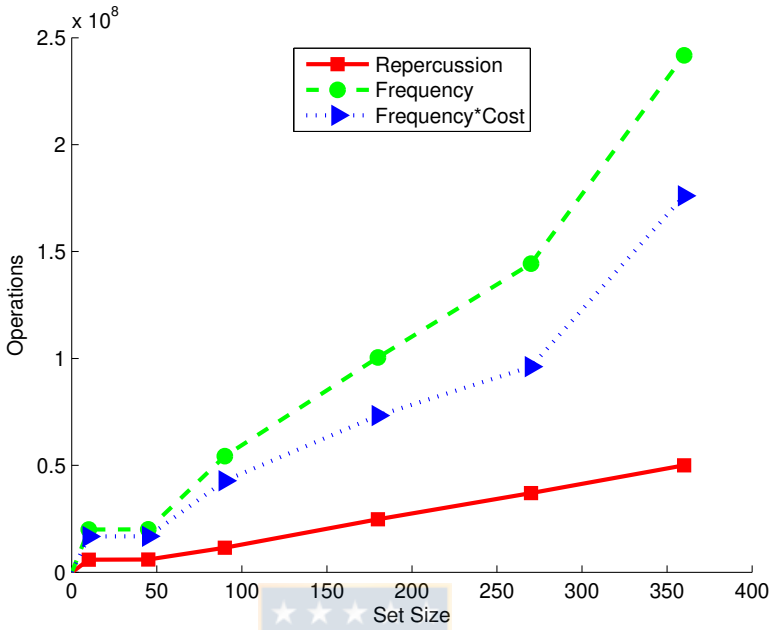

Figura 2.1: NIST Freeman Dataset. Operations vs Size of the Set

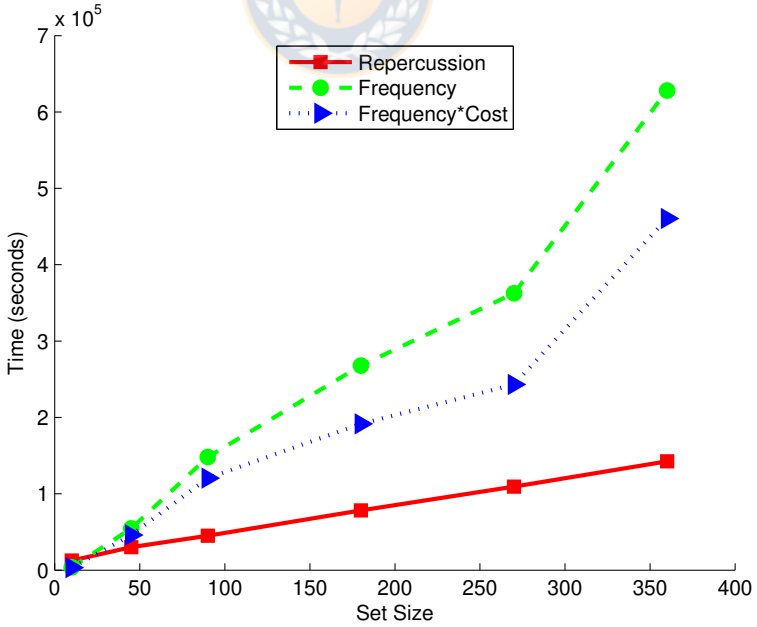

Figura 2.2: NIST Freeman Dataset. Time vs Size of the Set

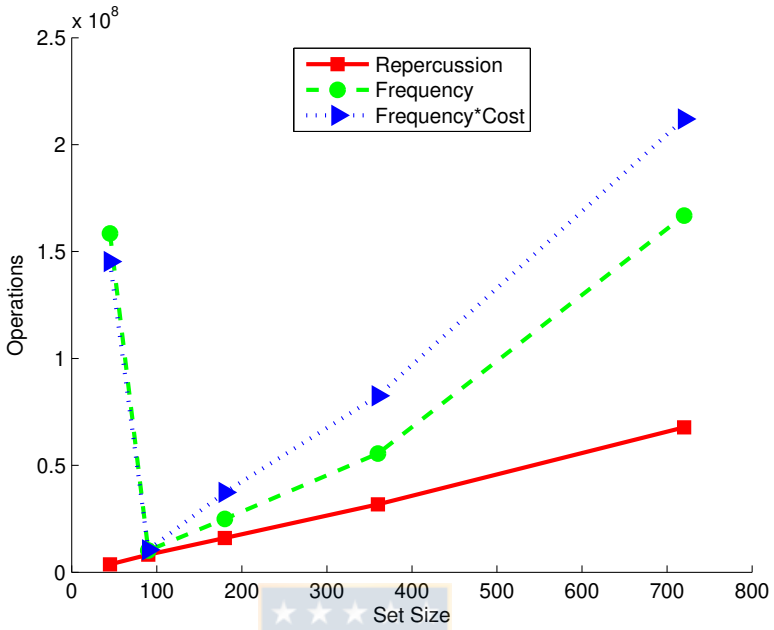

Figura 2.3: Proteins Dataset. Operations vs Size of the Set

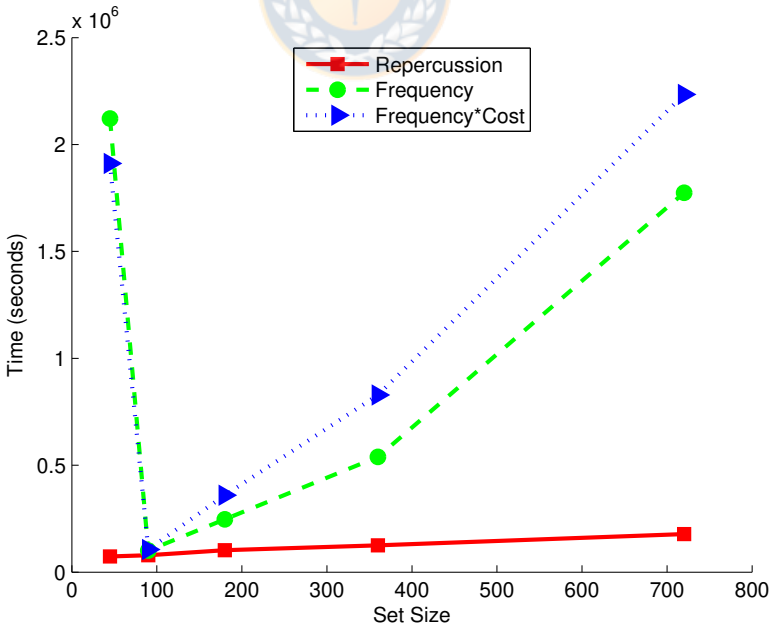

Figura 2.4: Proteins Dataset. Time vs Size of the Set

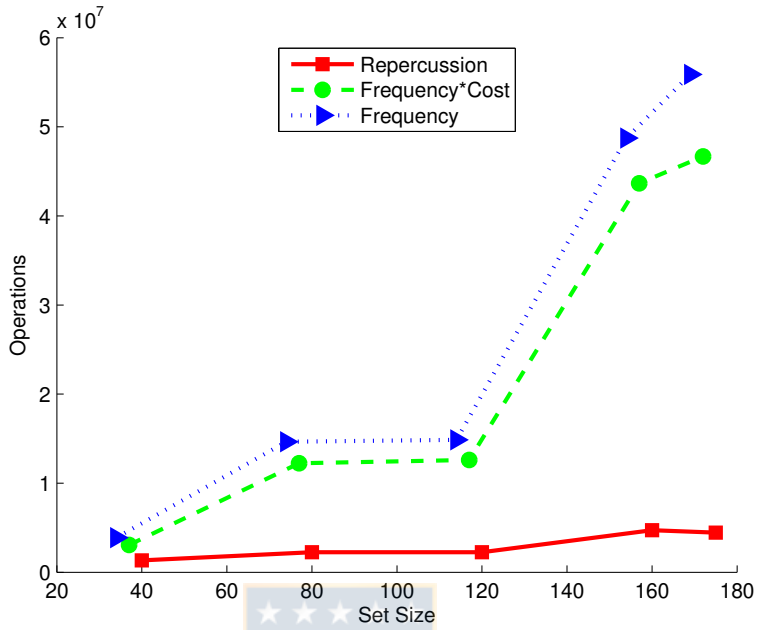

Figura 2.5: Insulin Dataset. Operations vs Size of the Set

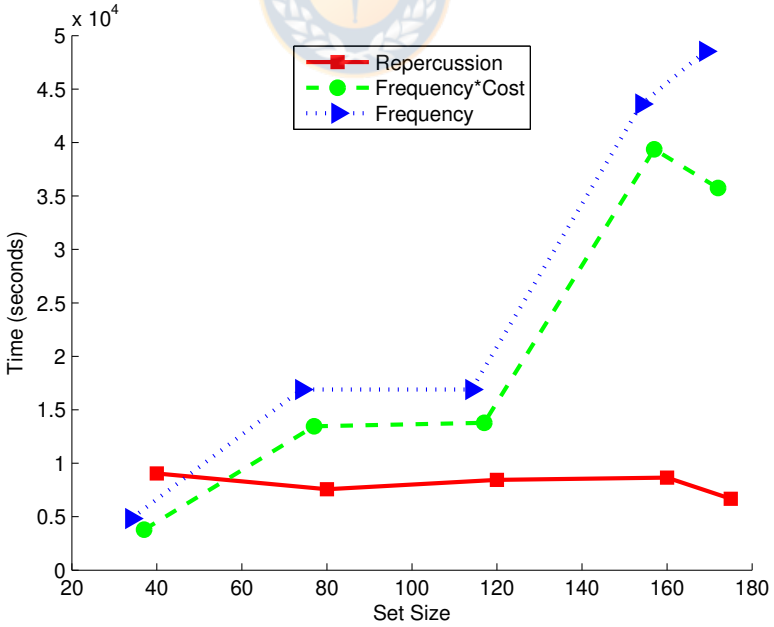

Figura 2.6: Insulin Dataset. Time vs Size of the Set

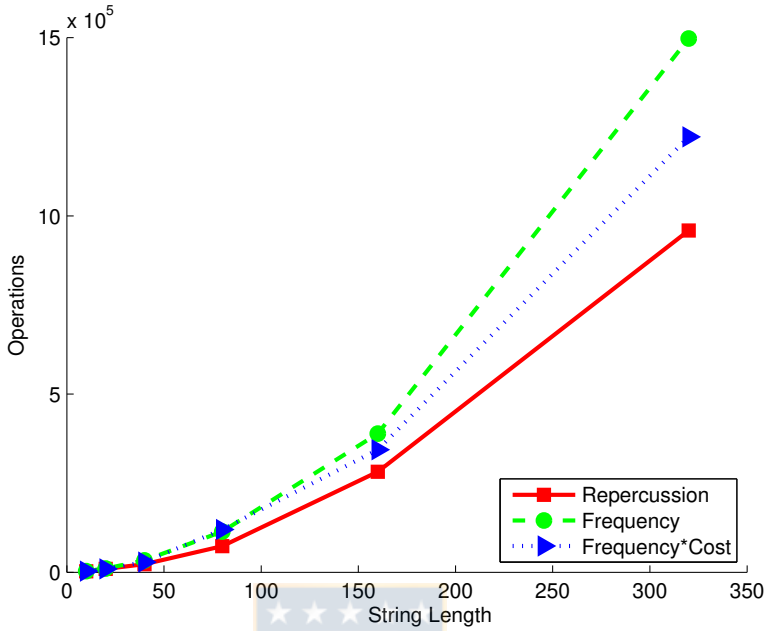

Figura 2.7: Synthetic Freeman Dataset. Operations vs Size of the Set

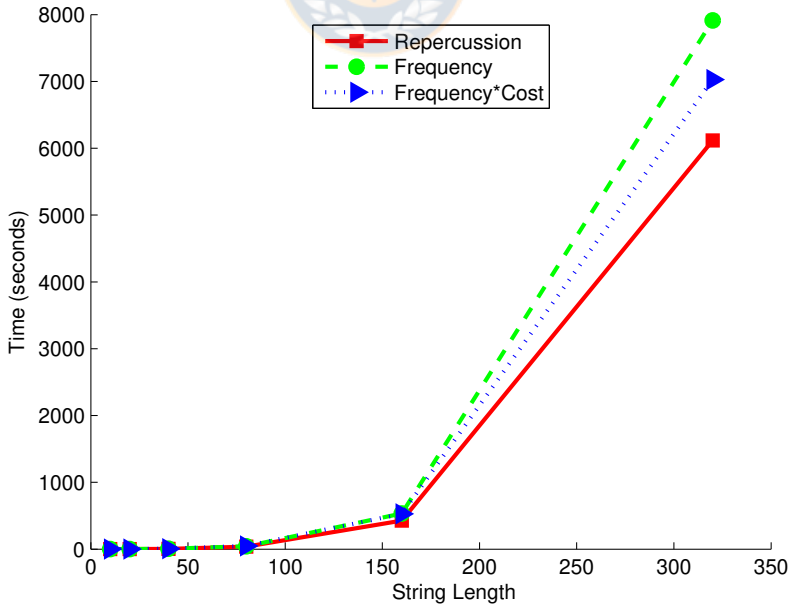

Figura 2.8: Synthetic Freeman Dataset. Time vs Size of the Set

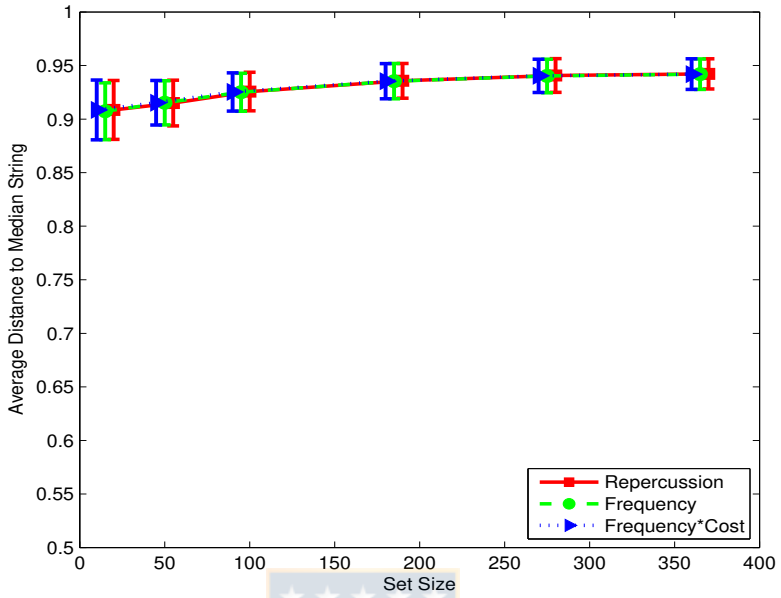

Figura 2.9: NIST Freeman: Median Quality

the other datasets, i.e. the algorithms in [44] obtained a median of similar quality to our proposal but requiring much more operations. Moreover, when the size of the set increases, algorithms in [44] perform fewer operations than expected, but this is probably caused by the early convergence to a local optimum.

Finally, we study how distance from the candidate median to each string in the set decreases with the number of iterations. We refer to this measure as the *error decrease*. This prevents misleading conclusions in cases of algorithms which results are obtained by a stagnation (or by a drastically decrease) in the last steps. In other words, it is desirable to provide algorithms that, not only require few operations to converge but also their convergence speed is fast. Fig. 2.13, Fig. 2.14, Fig. 2.15 and Fig. 2.16 show that our method has a great convergence speed.

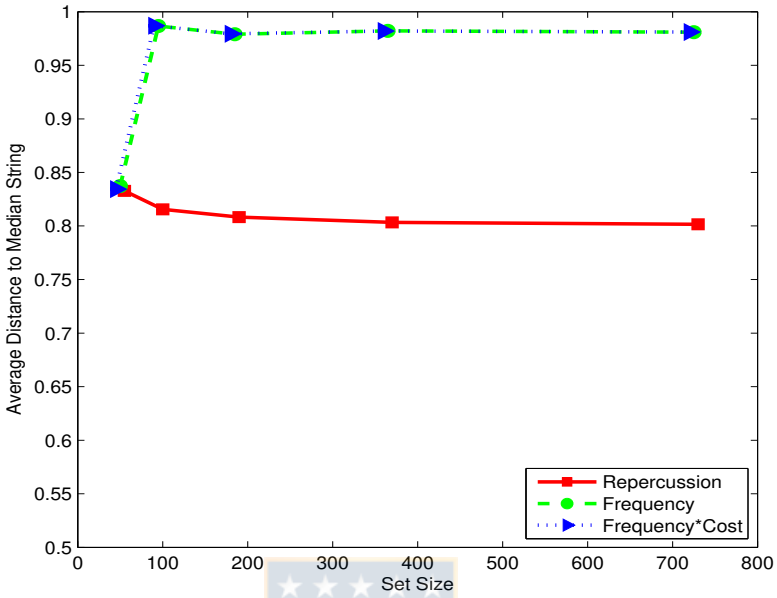

Figura 2.10: Proteins: Median Quality

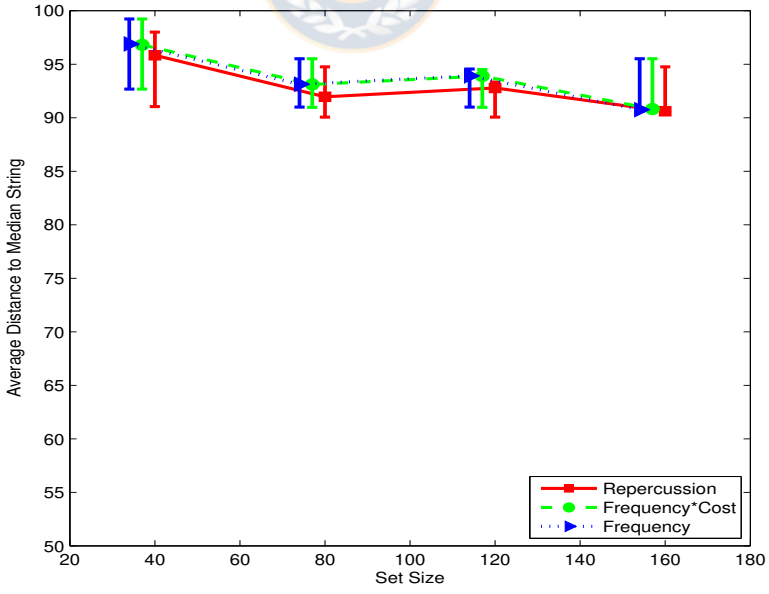

Figura 2.11: Insulin: Median Quality

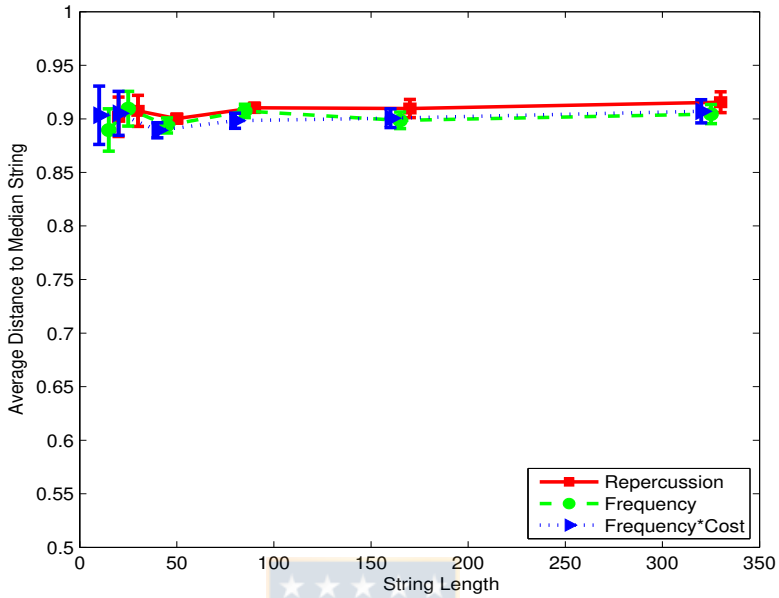

Figura 2.12: Synthetic Freeman: Median Quality

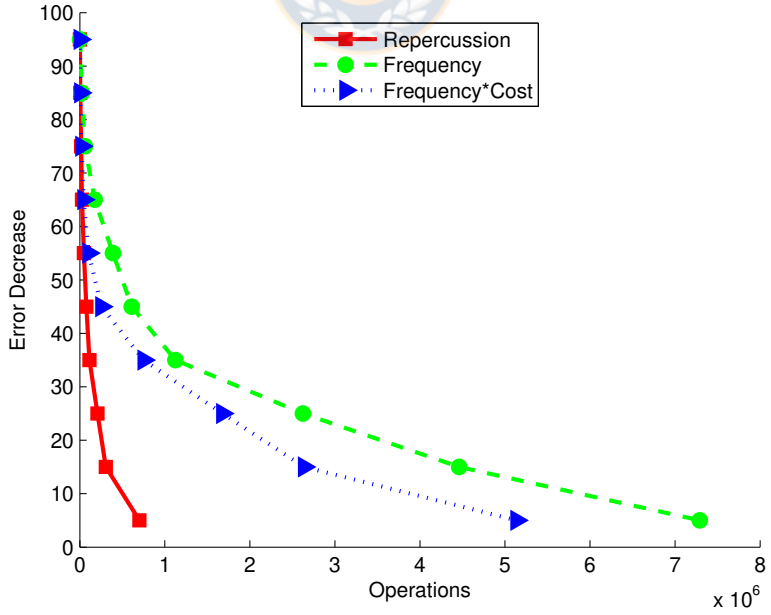

Figura 2.13: NIST Freeman: Error Decrease

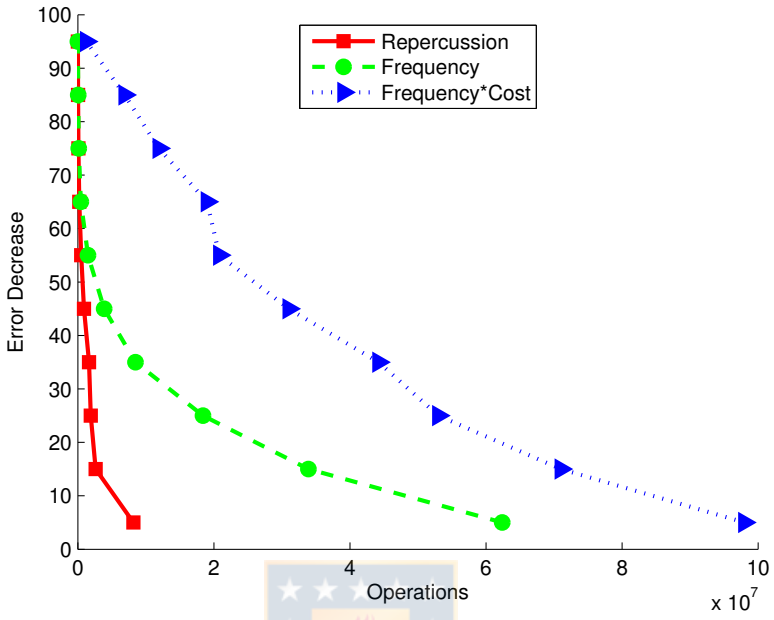

Figura 2.14: Proteins: Error Decrease

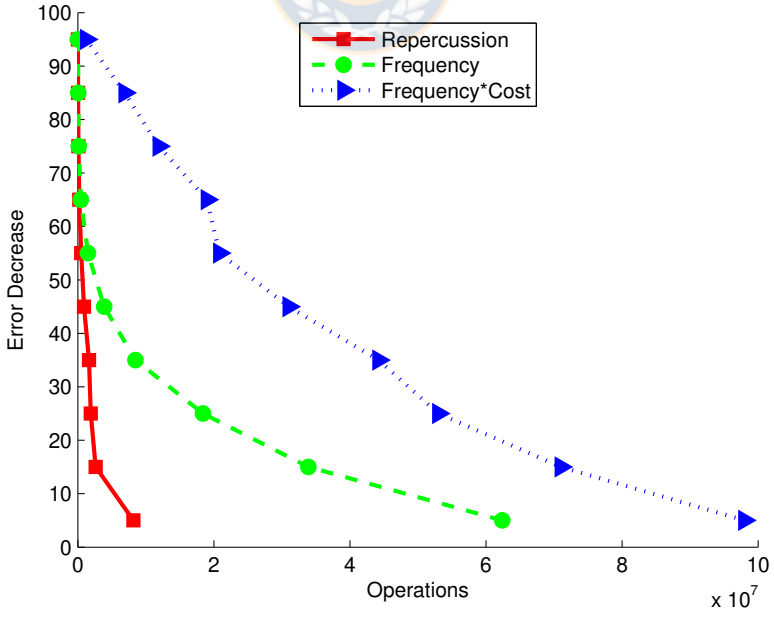

Figura 2.15: Insulin: Error Decrease

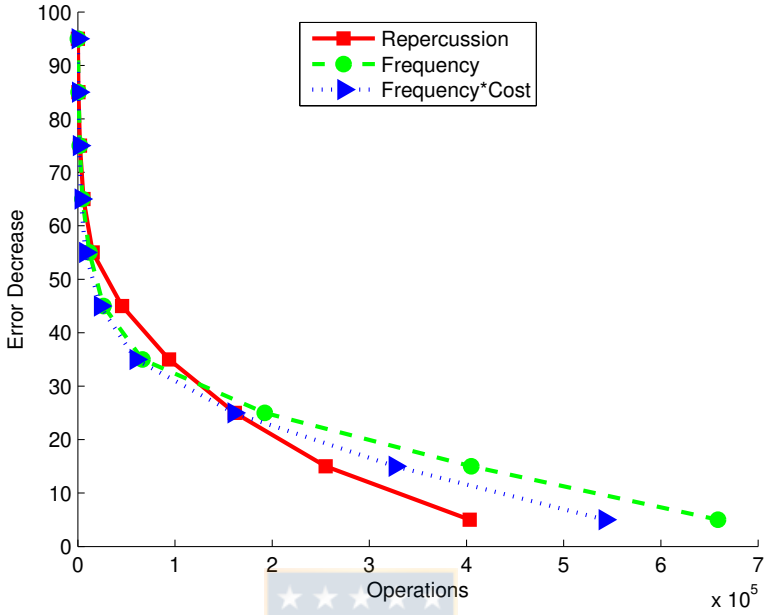

Figura 2.16: Synthetic Freeman: Error Decrease

2.1.7. Conclusions and Future work

In this paper, we present a novel approach to assess the best edit in perturbation-based iterative refinement algorithms to compute the median string. Our main contribution is to consider not only the repercussion of an edit e_k when it is part of the optimal edit sequence but also when it is not. We take this into consideration to propose a ranking of operations that, in practice, outperforms state of the art heuristic approximations to the median string in terms of convergence speed and quality of the approximated median computed. These conclusions are supported by an experimental evaluation with four datasets of different nature and characteristics.

Future research may compare these and other methods in terms of robustness to local optimums. Besides, it may be interesting to study strategies that, searching only on a subset of the strings,

lead to a quality approximations to the median string of the whole set. Also how to analyze the repercussion when multiple edits are applied at the same time.

Acknowledgements

This paper is funded in part by CONICYT through a Ph.D. Scholarship number 63140074; the Universidad Católica de la Santísima Concepción through the research project DIN-01/2016; European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement 690941; Millennium Institute for Foundational Research on Data (IMFD); and Fondecyt-Conicyt grant number 1170497.

2.2. Pivot Selection for Median String Problem.

Título: *Pivot Selection for Median String Problem.*

2.2.1. Abstract.

The Median String Problem is NP-Hard under the Levenshtein Distance, thus in practice, approximation heuristics are used. Perturbation based heuristics have been proved to be very competitive as regards of the ratio approximation accuracy/convergence speed. However, the computational burden increase with the size of the set. In this paper, we explore the idea of reducing the size of the problem by selecting a subset of representative elements, which are used to compute the approximate median string instead of the whole set. We aim to reduce the computation time by means of reducing the problem size while achieving similar approximation accuracy. We refer to those elements as pivots. We explain how we managed to find those pivots and how to compute the median string from them. Results on commonly used test data suggest that our approach can reduce the computational requirements (measured in computed edit distances) by 8 % with approximation accuracy as good as the state of the art heuristic.

2.2.2. Introduction

Different pattern recognition techniques such as *clustering*, classifiers *k-Nearest Neighbors* (kNN) or instance reduction algorithms require prototypes to represent different patterns. Strings are widely used to represent those patterns. One particular case is contour representation using eight-directions Freeman chain codes. Also,

strings are used to represent biological data such as DNA, RNA or protein sequences. The problem of finding a string that represents a set of strings has direct applications in relative compression algorithms and can be used with data such those mention previously. Finding the median string is an NP-Hard under Levenshtein distance. As an alternative to this, heuristic approaches have been proposed that iteratively refine an initial solution, applying editing operations until an approximation of the real median string is reached. However, as the size of the data sets begins to grow, the time of heuristics to find a solution to the problem increases. For that reason, focusing the search on certain elements can be fundamental to reduce the computational effort.

The Median String problem can be formalized as follows: Let Σ be an alphabet, Σ^* the set of all strings over Σ, and ϵ the empty symbol over this alphabet. For two strings S_i, $S_j \in \Sigma^*$, we denote as $E_{S_i}^{S_j} = \{e_1, e_2, ..., e_n\}$ the sequence of edit operations transforming S_i into S_j. Also, let $\omega(a \rightarrow b)$ be a domain specific function that assigns a cost to an edit operation. The cost of $E_{S_i}^{S_j}$ is $\omega(E) = \sum_{e_i \in E} \omega(e_i)$ and the edit distance from S_i to S_j is defined as $d(S_i, S_j) = argmin_E\{\omega(E_{S_i}^{S_j})\}$. The *median string* is a string that minimizes the sum of distances to all elements in the set. Neither the median string, nor the set median have to be unique.

2.2.3. Related Work

It is a fact that we can compute the median string using the Levenshtein distance for a set of N strings of length l in $\mathcal{O}(l^N)$ [17]. However, this computational cost is impractical. As an alternative, one approach consists in taking an initial string and makes successive editions over it, aiming to reduce the cumulative distance to the strings in the set. This family of algorithms are called

Perturbation-based algorithms.

The most frequent choices for the initial string are the empty string and the median of the set. A ranking of possible editions allows sorting them to be applied, the way this ranking is implemented affects the computational effort. In [31] is described a greedy implementation and in [47] is included a tie-breaking criterion for instances where many symbols have the same goodness index.

In [5], the current solution is systematically changed, performing insertions, deletions, and substitutions in every position, taking as the initial string the set median. Martínez et.al. [42] use a specific order to apply operations. First, they perform substitutions in each position of the candidate solution, evaluating each possible symbol.

Every time that a new candidate solution is generated, it is necessary to compute the distance to all the strings in the set. This happens in all the cases cited above. In [14, 43], authors apply multiple perturbations in the same iteration. These algorithms are faster, but the quality of the approximated median that they obtain is lower.

In [6] is studied how to rank each candidate edition, for apply first the bests. [44] improved the idea of [6] and achieved more solid outcomes, increasing the convergence speed compared with [48] and maintaining the quality of the approximated median. One step further, in [45], the heuristic to select the best edit operation considerate the repercussions of each edition in all strings of the set, improving the ranking of editions.

Other approach for the Median String Problem is as a Linear Programming problem, this is the case of [38], that give a lower bound and analyze the cases where the true median cannot be achieved, but do not obtain the median string. Also, [39, 40], using Integer Linear Programming, provide models for median and cen-

ter string problems. In [6], the weighted median string is presented as a generalization of this problem. In the case of, [37], strings are embedded in a vector space, but this approach has constrictions in terms of length variation and the maximum edit distances among the set.

In general, the whole input set S is used when running the algorithms. Perturbation-based heuristics use the set median or the empty string as the starting element. In this work, we propose to search the approximate median, not over S but in a set P such that $|P| \leq |S|$. This way, we expect to speed up the convergence of a reference perturbation-based heuristic while achieving similar approximation accuracy.

In this work, we explored two variations to the perturbation-based heuristic algorithm for the median string problem in [45]. When computing the median of a set S, we defined the starting string as the median of the set P of the pivots, instead of the median of S as in previous works, note that $|P| \leq |S|$. Also, we run the algorithm in [45] over P instead of S. However, the accuracy of the approximation is evaluated using S.

2.2.4. Reducing the problem. Search in Metric Spaces and Pivot Selection

A simple definition of a *Metric Space* is as follows:

A *Metric Space* is a set of elements in which the distance between them satisfies four basics rules. First, the distance between any pair of elements is not negative. Second, the distance between two elements x and y is *0* if and only if x and y are the same element. Third, symmetry, the distance from x to y is the same that the distance from y to x. And fourth, triangle inequality, distance

from x to z is always greater or equal to the sum of the distances from x to y and the distance from y to z.

1. $d(x, y) \geq 0$
2. $d(x, y) = 0 \leftrightarrow x = y$
3. $d(x, y) = d(y, x)$
4. $d(x, z) \leq d(x, y) + d(x, y)$

In this paper we use *Levenshtein Distance* as edit distance, which satisfies the properties of *Metric Space*.

It is very common in Computer Science problems trying to find the elements belonging to a set that have similarity, according to a specific function, greater than a certain threshold. The similarity function may be extremely complex in some cases, that is why to reduce the search space is a valid approach to decrease the search time.

One approach to this problem is focused on finding representatives of each region of the search space, these representatives are called pivots. When querying for the top-k similarity of a new element, the search space is reduced only to the neighborhood of the closest representative. The number of pivots varies depending on the accuracy of the similarity estimation. Normally, it is desired to have a balance between the number of pivots and the accuracy of the similarity estimation [59, 60], because a large number of pivots reduces the speed of the initial search, moreover, few pivots reduce the accuracy because in this case each pivot will represent several elements.

In [61] is widely described the process of Sparse Pivot Selection and remarks the previous work in this field like in [62], [63], [64]

and [65]. As we can see in the aforementioned works, the robustness of the similarity search method based on pivots depends directly on the number of pivots, the distribution between them and the distribution in the metric space. Precisely in [64] the maximization of the distance between pivots is pursued, showing empirical results of the effectiveness of this method. More recently [66] presents a dynamic method of pivots selection that can modify the set of pivots while the database is growing.

Our first approach was to extend the concept Pivot Selection in Metric Spaces to the median string problem presented in [61]. The idea is to reduce the number of distances calculated by testing different algorithms. All these algorithms have in common the estimation of the max distance in the set and the presence of an α value that regulates the number of pivots.

Algorithms for Pivot Selection

As we described in Section 2.2.4, the robustness of the similarity search method based on pivots depends directly on the number of pivots, the distribution between them and the distribution in the metric space. A good pivot selection can reduce the search space. That is why, an estimation of the maximum distance between elements is required, and for this, we use a linear algorithm. The idea of that algorithm is to calculate the maximum distance from the first element to the rest of the set, and then, do the same with the farthest element found, keeping a reference of the maximum obtained distance. This process ends when no improvement is achieved. This idea is illustrated in Algorithm 2.

In order to properly select the pivots, a parameter α is needed,

Input : S
Output : $maxValue$

```
1  currentIndex = 0
2  formerIndex = −1
3  maxPossition = 0
4  maxValue = −∞
5  while (currentIndex ≠ formerIndex) do
6      foreach (i ∈ S) do
7          dist = getDistance(S(i), S(currentIndex))
8          if (dist > maxValue)
9          then
10             maxValue = dist
11             maxPossition = i
12     formerIndex = currentIndex
13     currentIndex = maxPossition
14 return maxValue
```

Algorithm 2: maxDistanceEstimation(S) :$maxValue$

which is a value between 0 and 1. This value represents a fraction of the maximum distance in the set. We also want to know how many set members are represented for each pivot. Algorithms 3 shows our procedure to select pivots.

Input : $S, \alpha, maxValue, setMean$
Output : P, W

```
/* P: List of pivots .                                          */
/* W: List of respective pivot weight.                          */
P = ∅
W = ∅
/* setMean is the first pivot and its weight is 1 .             */
P.add(setMean)
W.add(1)
foreach (i ∈ S) do
    posible = true
    minSpace = ∞
    pivotIndex = 0
    foreach (j ∈ P) do
        space = getDistance(P(j), S(i))
        if (space < maxValue * α) then
            posible = false
            if (space < minSpace) then
                minSpace = space; pivotIndex = j;
    if (posible) then
        P.add(S(i))
        W.add(1)
    else
        W(pivotIndex)++
return P, W
```

Algorithm 3: pivotSelection$(S, \alpha, maxValue, setMean)$ $:P, W$

To adapt the pivot selection algorithm to the median string problem we propose Algorithm 4. AppMedianStringRepercussion

refers to the algorithm proposed in [45]

Input : S, α
Output : $\hat{P}$

1 $meanS = getMean(S)$
2 $maxDistance = maxDistanceEstimation(S)$
3 $[P, W] = pivotSelection(S, \alpha, maxDistance, meanS)$
4 $\hat{P} = AppMedianStringRepercussion(P, getMean(P))$
5 **return** $\hat{P}$

Algorithm 4: AppMedianStringRepercussionP(S,α) :$\hat{P}$

During the maximum distance estimation and pivot selection we count the number of distances computed for comparative reasons. We add it to the result of median string distance count. Also the median string resultant quality is calculated using all S members and not only the pivots.

2.2.5. Experimental Results

For experimental evaluation, we work with strings that represent contours of letters using Freeman Chain Codes of Eight Directions, from the *NIST-3 Uppercase Letters* database. This codification is also used in [54, 55, 56, 44, 45]. For Freeman Chain Codes, substitution cost between symbols is equivalent to one unit for every 45 degrees of difference in the orientation of each symbol, for insertions and deletions the cost is always of two units as in [56, 44, 45]. We evaluate sets of size of 360 strings, for each letter of the English alphabet, 26 independent samples were drawn.

We classify the *NIST-3 Uppercase Letters* datasets, according with the average string length, in short, medium and large. For these experiments we selected some datasets from each class. For short length datasets we used letters P, O and I, for medium length

datasets we used R, D, B and A, and for large length datasets we used W and M. For each one, we compare results using different α values. We show the number of operations required (in thousands), the average distance to the median (MAD) and the percentage of pivots (Pivots %) with respect to the whole set. We also show in the last columns of each table, from Table 2.4 to Table 2.12, the *Reference Result* obtained without the pivot selection strategy. For the maximum distance estimation and pivot selection, we count the number of distances computed for comparative reasons. We add it to the result of median string distance count. Also the median string resultant quality is calculated using all S members and not only the pivots.

α	0.15	0.13	0.12	0.11	Reference
Pivots %	68.61	84.72	90.00	97.22	Result:
Operations	653	819	794	872	910
MAD	91.31	90.16	89.92	89.73	89.84

Cuadro 2.4: Median String Dataset P.

α	0.17	0.16	0.15	0.10	Reference
Pivots %	76.94	81.11	91.11	99.72	Result :
Operations	1239	1551	1223	1664	1598
MAD	132.54	132.19	131.35	131.19	131.40

Cuadro 2.5: Median String Dataset R.

Analyzing the results we find that when Pivots % decreases the MAD increases. Also, if Pivots % grows near to 100, the number of operations can be even larger than the obtained without the pivot

α	0.20	0.18	0.15	Reference
Pivots %	61.39	75.83	93.33	Result :
Operations	3267	3843	3675	4044
MAD	218.12	215.36	213.64	214.13

Cuadro 2.6: Median String Dataset W.

α	0.15	0.12	0.11	0.10	Reference
Pivots %	47.78	77.22	86.11	91.94	Result :
Operations	324942	594061	538963	626987	536123
MAD	67.46	63.05	62.50	62.06	62.20

Cuadro 2.7: Median String Dataset O.

α	0.20	0.15	0.1	0.07	0.05	0.04	Reference
Pivots %	21.67	30.83	39.17	58.61	82.22	93.33	Result :
Operations	462	606	877	910	786	1026	1577
MAD	200.64	185.41	165.97	124.55	110.38	109.20	109.46

Cuadro 2.8: Median String Dataset I.

α	0.20	0.15	0.14	0.13	0.12	0.1	Reference
Pivots %	35.83	64.16	70.28	82.78	88.3	97.5	Result :
Operations	1669	1285	1720	1155	1293	1440	1165
MAD	124.31	118.64	117.57	116.61	116.40	115.97	116.29

Cuadro 2.9: Median String Dataset B.

selection strategy. These experiments show that the best results are achieved when the percentage of pivots is between 85 % and 95 % because the MAD is very close to the MAD for the Reference Result

α	0.30	0.20	0.15	Reference
Pivots %	29.17	81.94	97.5	Result :
Operations	365	807	695	928
MAD	104.24	96.98	96.42	96.35

Cuadro 2.10: Median String Dataset A.

α	0.15	0.12	0.10	Reference
Pivots %	57.78	81.94	94.72	Result :
Operations	840	691	809	729
MAD	102.04	98.01	97.36	97.64

Cuadro 2.11: Median String Dataset D.

α	0.20	0.19	0.185	0.18	0.17	0.15	Reference
Pivots %	70.83	76.67	81.39	85.00	90.00	95.56	Result :
Operations	2696	2256	2397	2398	2647	2478	2363
MAD	209.44	207.30	207.05	206.50	205.61	205.33	205.83

Cuadro 2.12: Median String Dataset M.

while the number of operations that we need is smaller.

2.2.6. Conclusions

In this work, we explored two variations to the perturbation-based heuristic algorithm for the median string problem in [45]. When computing the median of a set S, we defined the starting string as the median of the set P of the pivots, instead of the median of S as in previous works, note that $|P| \leq |S|$. Also, we run the algorithm in [45] over P instead of S. However, the accuracy of

the approximation is evaluated using S.

Since Pivots % depends on α we test different values for this parameter. In general, as α increases, Pivots % and the number of operations decreases, but the MAD increases. However, if we look at values of Pivots % for those the MAD is not worst than the MAD for Reference (except the fractional part) we note that our approach can reduce the number of operation in 8 % as average. For example, for letter P, the reference MAD is 89,84 and the number of operations $910K$. For $\alpha = 0{,}12$, we had that Pivots % is 90,00 the MAD is 89,92 and the number of operations $794K$ which is a reduction of 12,7 %

Results also suggest that the value α that can lead to a reduction of the number of operations without affecting the MAD is different in each data set. We would like to explore an approach to determine a α that is good enough for all data sets.

Besides, we will try different ways of applying the Pivot Selection to the Median String Algorithms. For instance, we will try to use Pivot Selection to get the initial string for our median string algorithm.

2.2.7. Acknowledgments

This paper is funded in part by CONICYT-PCHA / Doctorado Nacional / 2014 - 63140074 through a Ph.D. Scholarship; the Universidad Católica de la Santísima Concepción through the research project DIN-01/2016; European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement 690941; Millennium Institute for Foundational Research on Data (IMFD); FONDECYT-CONICYT grant number 1170497; and for O. Pedreira, Xunta de Galicia / FEDER-UE refs. CSI ED431G / 01 and GRC: ED431C 2017/58.

2.3. Boosting the median string algorithms

Título: *Boosting perturbation-based iterative algorithms to compute the median string.*

2.3.1. Abstract.

The median string problem is NP-hard under several formulations. The most competitive heuristics use perturbation-based iterative algorithms. The starting string and the policy to order possible edit operations are central to the efficiency of such approaches. In this paper we attack both sub-problems. First, for the starting point we use the median string of a small subset of the input set. These strings are the Half Space Proximal (HSP) neighbors of the median of the set. The HSP neighbors are simultaneously close to the center and are diverse among them. Second, we show how to manage the list of possible editions to trim the list of candidate operations and reduce the search space for achieving a faster convergence. To validate these results, we present comparative experiments, attending mainly to the quality of the median obtained, the time to compute such median, and the number of edit distances computed.

2.3.2. Introduction

As defined by [5] the *median* of a set S of strings is a string, not necessarily in S, that minimizes the sum of the distances to each element in the collection.The median string is often used in different contexts as a representative for a collection of objects. There are relevant fields where processing sequences of symbols is a key problem. For example in molecular biology for sequence alignment

and homology search [17], where improvements in sequencing techniques can generate large datasets that need to be processed. The median string is also used in applications such as prototype construction for 2D shape representations [7, 6], the clustering of strings [9], or Self-Organized Maps of strings [13, 14, 15]. Algorithms for relative compression of genomes, such as [20, 21], are another field where finding such string has been receiving increasing attention. Compression is achieved by analyzing commonalities of sequences relative to a reference string R. Thus, selecting R is a key issue as point out [22, 23, 24]. Other problem that requires the analysis of discrete symbol sequences is the Anomaly Detection with application in Airline Safety [67].

Computing the median string using Levenshtein Distance is a problem within the NP-Complete class, as shown in [31, 33], several approaches have been proposed, such as the one called perturbation-based iterative refinement [5, 41, 14, 42, 43, 44, 45]. In this approach, perturbations are made to an initial string to improve the quality of the partial solution found, until reaching a point where no improvements are possible. A crucial part of this approach is the selection of the starting point. The best existing algorithms use the empty string or the median of the set as the initial string to be improved. So far the best results have been obtained by starting from the median of the set [45]. In this paper, we propose a new starting point, calculated from certain neighbors of the median of the set. On the other hand, a better view on the goodness score of the candidate perturbations has led us to determine a point at which the low probability of improvement of remaining candidate operations does not compensate the computational effort required, achieving a reduction in the number of edit distance calculations necessary.

2.3.3. Preliminaries and Related Work.

We formalize the Median String problem with the same notation used in [44], and [45]. Let Σ be an alphabet, Σ^* the set of all strings over Σ, and ϵ the empty symbol over this alphabet. For two strings S_i, $S_j \in \Sigma^*$, an edit operation is a pair $(a, b) \neq (\epsilon, \epsilon)$, written $a \to b$, which transforms a string S_i into S_j, if $S_i = \sigma a \tau$ and $S_j = \sigma b \tau$, where σ and τ represent substrings. To denote substitutions, deletions and insertions we use $a \to b$, $a \to \epsilon$, and $\epsilon \to b$, respectively. We denote as $E_{S_i}^{S_j} = \{e_1, e_2, ..., e_n\}$ the sequence of edit operations transforming S_i into S_j. Also, let $\omega(a \to b)$ be a domain specific function that assigns a cost to an edit operation. The cost of $E_{S_i}^{S_j}$ is $\omega(E) = \sum_{e_i \in E} \omega(e_i)$ and the edit distance from S_i to S_j is defined as $d(S_i, S_j) = argmin_E\{\omega(E_{S_i}^{S_j})\}$.

The *median string* is a string that minimizes $\sum_{S_i \in S} d(\hat{S}, S_i) | \hat{S} \in \Sigma^*$. A common approximation to the true median string is the *set median*, a string in S that minimizes $\sum_{S_i \in S} d(\hat{S}, S_i) | \hat{S} \in S$. Neither the median string, nor the set median have to be unique.

Early works of [17] show that it is possible to compute the median string in $\mathcal{O}(l^N)$ using the Levenshtein metric for a set of N strings of length l. This computational cost is impractical and different heuristics have been proposed since then. A common approach for median string algorithms takes an initial string and makes successive editions over it to reduce the cumulative distance to the strings in the set. We call this family of algorithms Perturbation-based algorithms. Methods in this family mainly differ on the initial string used and on the policy to perform editions.

The initial string varies depending on the authors, but the most frequent cases are the empty string and the median of the set. Trying all possible edit operations to transform and improve the initial string would be too expensive. Hence, to decide the edition

to be applied, a ranking of possible editions is used. [31] described a greedy implementation and [47] a tie-breaking criterion for instances where many symbols have the same goodness index.

In [5], the current solution is systematically changed, by performing insertions, deletions, and substitutions in every position, taking as the initial string the set median. Edits are only accepted if they lead to an improvement. [42] use a specific order to apply operations. First, they perform substitutions in each position of the candidate solution, evaluating each possible symbol. The new partial solution $\hat{S}^t$ is the string, selected from all the new candidates, that minimizes $\sum_{S_i \in S} d(\hat{S}^t, S_i)$. A similar procedure is repeated for deletions, substitutions, and insertions, in such order and until there are no more improvements. Let $\hat{S}^{t-1}$ be the approximated median at step $t-1$. Another alternative is to generate a number of candidate approximations from $\hat{S}^{t-1}$ by separately applying substitutions, deletions and insertions only at position i, starting with i at the first position of $\hat{S}^{t-1}$. The candidate with lowest $\sum_{S_i \in S} d(\hat{S}^t, S_i)$ replaces $\hat{S}^{t-1}$ for a new iteration in case of improvement. If no such a sequence exists, $\hat{S}^{t-1}$ remains as the best approximation found and the algorithm explores in the next iteration $i+1$. The search stops when there are no improvements. This approach achieves good approximations to the true median string.

In the approaches discussed above, every time an edition is applied to $\hat{S}^{t-1}$, it is necessary to compute the distance to all the strings in the set. No a priori ranking of operations is provided, so a goodness criteria is needed to discriminate between possible editions in order to speed up the algorithms. In that sense, [48] prefer some operations among others, for example, substitutions only are evaluated for the two closest symbols to the examined position. In [14], and [43], authors apply multiple perturbations in the same ite-

ration. These algorithms are faster, but results in [44] suggest that the quality of the approximated median that they obtain is lower.

[6] studied how to score the goodness of each candidate perturbation in order to evaluate first the most promissory ones. [44] refined the idea of [6] and obtained better results, improving the convergence speed in comparison with [48] while preserve the quality of the approximated median. One step further, in [45], the heuristic to select the best edit operation takes into consideration the repercussions of each edition e_k not only in strings for whom $e_k \in E_{Sk}^{\hat{S^{t-1}}}$ but in all strings of the set, improving the goodness index for the editions.

The Median String Problem has been also studied from other perspectives, such as Linear Programming. [38] provide a lower bound and analyze the cases where the true median cannot be achieved. Unfortunately, they do not compute the median string. Also, [39, 40], using Integer Linear Programming, developed models for median and center string problems, considering special cases when the distance between the strings in the set is bounded and the dissimilarity measure satisfies the triangle inequality. In some cases, the search for the median string relies on the weighted median concept described in [6]. This is the case of [37], that embedded the strings in a vectorial space. This approach has constrictions in terms of the variation of the set both in length variation and the maximum edit distances among the set. A different approach based on an evolutionary framework was proposed by [51]. Finally, the median string has been studied for other metrics different from Levenshtein edit distance. For example, [49] studied the approximation to the median string under the stochastic edit distance.

2.3.4. Our Proposal

As explained in previous section, in perturbation-based iterative refinement algorithms, perturbations are applied to an initial string to improve the quality of the partial solution found, until reaching a point where no improvements are obtained. We hypothesized that a better starting point can reduce the number of edit distances that have to be calculated to obtain the median string. Authors like [14] and [44] experimented with taking the empty string as their starting point. Also, authors such as [5, 31, 48, 44, 45] obtained their best results when taking the median of the set as starting point. Their experiments showed that the starting point has an impact on the performance of the algorithms.

Our approach aims at obtaining a sample of the string set with two simultaneous properties, namely *diversity* and *proximity*. Both goals can be achieved using the Half Space Proximal(HSP) graph, defined originally in [46] and used also to learn a representation of the foldspace of proteins in [68]. The HSP graph is a sparse subgraph of the complete graph in a metric space. The HSP graph of a set S is constructed by independently applying the *HSP test* to each object in $S_i \in S$. The output of the HSP test on S_i is a partition of S, $\mathcal{P} = \{P_1, P_2, ...\}$, where each P_j has a representative object p_{jr}, that is connected to S_i in the HSP graph.

In our proposal we do not need to construct the complete HSP graph. We only apply the HSP test to the set median m of S. Therefore, the HSP test takes two parameters, the set of strings S, and its median m (which we assume has been previously computed). The HSP test proceeds iteratively. Let S' be the set of strings in the iteration i (when $i = 0$, $S' = S$). In each iteration i, the algorithm finds the nearest neighbor of m in S', $p_{ir} = kNN(m, S')$. All the objects more similar to p_{ir} than to m form a new subset

P_i of S', where p_{ir} is the representative of P_i. All the objects in P_i are removed from S', and m will be connected in the HSP graph to the representative objects of the partition, p_ir. The algorithm proceeds to a new iteration, and finishes when S' is empty. A final step reassigns all the strings S' in each subset P_j to the subset corresponding to the closest representative of S'.

The result of applying the HSP test on m gives us a partition of S, $\mathcal{P} = \{P_1, P_2, ...\}$. The representatives of the elements of the partition form a new subset of S, $R = \{p_{1r}, p_{2r}, ...\}$. The objects in the HSP test S' of m fulfill the properties of diversity and proximity. With this new information, we recompute the initial string for the improvement process as the weighted median of the set S'. As we will see in the experimental results, this improves the performance of the algorithm by reducing the number of edit operations we have to perform in the initial string.

Algorithm 5 shows our implementation of Half Space Proximal test. This algorithm takes two parameters as input, a set S and an element m, which has been previously removed from S. The output of Algorithm 5 is a partition P of S into disjoint subsets P_i. Each of the sets $P_i \in P$ has a representative p_{ir}. Each p_{ir} is the closest element to m within P_i. Initially, P is the empty set (line 1). The first part of Algorithm 5 (lines 2-12), iterates until each element in S is added to some subset $P_i \in P$. First, an empty set Q is created. We find the element $nn \in S$ closest to m and add it to Q (lines 4-7). All the elements added to Q are removed from S. For all the elements $S_i \in S$, if the distance between S_i and nn is smaller than the distance between S_i and m, the element S_i is added to Q and removed from S (lines 8-11). At the end of each iteration of this part of the algorithm, Q is added to P (line 12). In the second part of Algorithm 5 (lines 13-24), each element $p_{ij} \in P_i$, such that $p_{ij} \neq p_{ir}$

, is reassigned to another subset P_j in case the distance between p_{ij} and p_{jr} is less than the distance between p_{ij} and p_{ir}. The second part of the algorithm takes $\mathcal{O}((n-p) \times p)$ where n is the size of S and p is the number of subsets in P. As p of the n elements are representatives, we have $(n-p)$ non-representative elements. For each of the $(n-p)$ non-representatives, it is necessary to compare the distance to each representative p_{ir} in order to determine to which P_i it belongs.

In Fig. 2.17 to Fig. 2.21 we illustrate an example of the Half Space Proximal test. We show a possible spatial relation of the different strings in the set S in terms of distance between them. In Fig. 2.17, the star represents the median of the set $m \in S$. Then, in Fig. 2.18, string s_1 is selected as the closest to m, and space is split in two, any string closer to s_1 than to m is assigned to P_1 and $p_{1r} = s_1$. In Fig. 2.19, the string $s_2 \notin \bigcup_{i=1}^{n} P_i$, is selected as the closest to m and the previous process is repeated to build P_2. This process continues, selecting the nearest string s_3 and building P_3 and so on, until all strings become part of the set $P = \{P_1, P_2, ..., P_n\}$. In our example, we can see in Fig. 2.20 that all strings are already assigned. The last stage of the test is to reassign all strings $s_i \neq p_{ir}$ to the P_i with the p_{ir} closest to s_i. The final reassignment is shown in Fig. 2.21.

We propose Algorithm 6 for calculating the median string when strings in the dataset have different weights. This algorithm takes as input an instance set S, an initialization string R, and a weight set W that is a vector that represents the corresponding weight of each element in S. The algorithm iterates through the same steps until no editions applied to $\hat{S}$ improve the result. First, the distances between R' and each S_i and the respective involved editions $E_{S_i}^{R'}$ are computed (lines $4-6$). Each edition in $E_{S_i}^{R'}$ have an associated

Input : Set S, Set Median m
Output: Set of Sets P

```
1  P = ∅
2  while S ≠ ∅
3  do
4      Q = ∅
5      nn = argmin{distance(S, m)}
6      Q.add(nn)
7      S.remove(nn)
8      foreach S_i ∈ S do
9          if distance(S_i, nn) < distance(S_i, m) then
10             Q.add(S_i)
11             S.remove(S_i)
12     P.add(Q)
13 foreach P_i ∈ P do
14     foreach P_ij ∈ P_i; j > 1 do
15         minDistance = distance(P_ij, P_i0)
16         minP = i
17         foreach P_k ∈ P do
18             if distance(P_ij, P_k0) < minDistance then
19                 minDistance = distance(P_ij, P_k0)
20                 minP = k
21         if minP ≠ i then
22             P_minP.add(P_ij)
23             P_i.remove(P_ij)
24 return P
```

Algorithm 5: HSP test pseudo-code.

weight W_i used to update the statistics (line 6). In lines $7-12$, the repercussion of each edition affecting the same position is computed, this generates a goodness index of editions. All editions are inserted in a priority queue Q, sorted by goodness index. Then, we discard from Q all editions q_i with $q_i.goodnessIndex \leq 0$ (line 16). Next, we dequeue editions from Q to obtain a new candidate R', applying

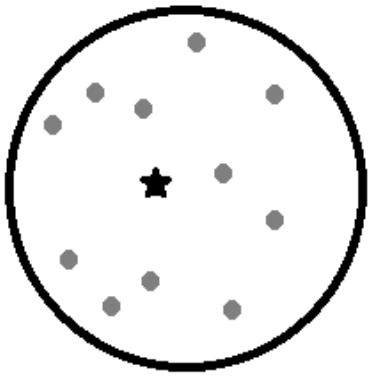

Figura 2.17: Starting point.

Figura 2.18: 1st neighbor.

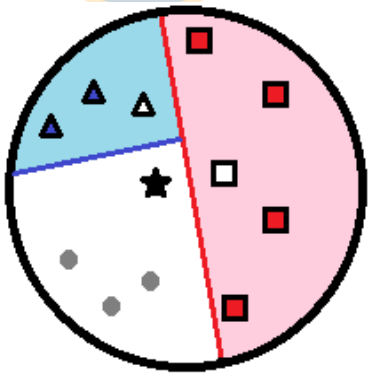

Figura 2.19: 2nd neighbor.

e_k to $\hat{S}$ (lines $13-15$), these two steps are repeated while the new candidate R' is worse than $\hat{S}$ and Q is not empty. Finally, the

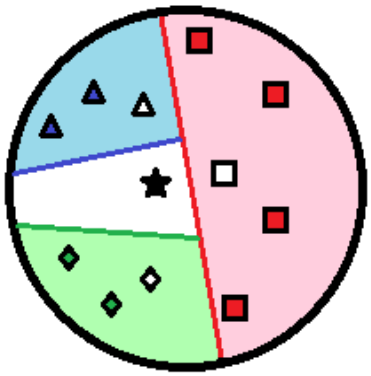

Figura 2.20: 3rd neighbor.

Figura 2.21: Reassignment.

algorithm returns $\hat{S}$.

This algorithm iterates by considering one perturbation at a time, until it does not get any improvement during the iteration. Each iteration may consider several different editions. In the worst case, this is upper bounded by $O(l \times \Sigma^2)$, where l is the length of the longest string. In the experimental evaluation, we show that this bound is rather pessimistic and our heuristic usually needs just a few editions per iteration. This is a key difference with the algorithm in [44], which uses more operations per iteration.

For each edition explored during an iteration, the algorithm computes the distance of the new candidate R' to all the elements

Input : instance set S, initialization string R, weight set W
Output: approximate median string $\hat{S}$

1 $R' = R$
2 **repeat**
3 $\quad \hat{S} = R'$
4 $\quad$ **foreach** $S_i \in S$ **do**
5 $\quad\quad$ compute $d(R', S_i)$ and the respective $E_{S_i}^{R'}$
6 $\quad\quad$ statistics.update($E_{S_i}^{R'}, W_i$)
7 $\quad$ **foreach** *position* j *of* R' **do**
8 $\quad\quad$ **foreach** *symbol* $si \in \Sigma$ **do**
9 $\quad\quad\quad$ **foreach** *symbol* $sj \in \Sigma$ **do**
10 $\quad\quad\quad\quad$ $Rep[j][si] += \omega(R[j] \rightarrow si) - \omega(si \rightarrow sj)$
11 $\quad$ Q: a priority queue of editions sorted by goodness index
12 $\quad$ Discard from Q all editions q_i with $q_i.goodnessIndex \leq 0$
13 $\quad$ **while** $\sum_{S_i \in S} d(\hat{S}, S_i) \leq \sum_{S_i \in S} d(R', S_i)$ **and** $Q \neq \emptyset$ **do**
14 $\quad\quad$ $e_k = Q.dequeue$
15 $\quad\quad$ obtain a new candidate R' applying e_k to $\hat{S}$
16 **until** *no editions* e_k *applied to* $\hat{S}$ *improve the result*
17 **return** $\hat{S}$

Algorithm 6: Median String Repercussion with weights.

in S (lines $17-20$), which takes time $O(N \times dc)$, where dc is the time to compute the edit distance (Levenshtein in our experiments). By providing a better ranking, we save on the number of operations explored per iteration, and thus, on the number of times this distance is computed, which is expensive (e.g., $O(l^2)$ for Levenshtein). However, to do that we expend some computations to bound the *repercussion* (lines $7-10$). This takes $O(l \times \Sigma^2)$ time and it is usually worth it as l and Σ are much smaller than N in most applications.

The algorithm presented by [45], generates a goodness index for each edition, taking into account how this edition impacts in

other editing alternatives affecting the same position. This goodness index is more precise than considering only the frequency of the editions or the frequency multiplied by the cost as in the case of [44]. In Algorithm 6, it is possible to see on line 15 that the operations are sorted by their goodness index, but, different from [44, 45], we propose to trim Q discarding those editing operations that have a negative value of goodness index, line 11. This modification can reduce the size of Q, speeding up the while loop in lines 13-25, because it runs until an improvement is achieved or Q is empty.

This algorithm is based on the Algorithm 1, presented in [45]. The original algorithm considered all strings $S_i \in S$ with the same relevance. We have made the necessary modifications so that strings $S_i \in S$ can have different weighs. It is important to notice that, unlike in [42, 48, 44, 45], for the calculation of the median of S' we use an approach in which each S'_i is weighted according to the size of the subset P_i that it represents. The idea of weighting strings has been studied in [36], but only when computing the median of two strings.

2.3.5. Experimental Results

Our experimental evaluation uses different alphabets, set sizes and string lengths. In Eq. 2.3, we show the ratio used to evaluate the quality of the obtained median string $\hat{S}$, where S^M is the set median. Besides, we compare the number of edit distances required by the algorithms to converge. Also, we took into consideration the execution time for each experiment. As expected, in all the experiments, time was proportional to the number of edit distances calculated.

$$\frac{\sum_{S_i \in S} d(\hat{S}, S_i)}{\sum_{S_i \in S} d(S^M, S_i)} \tag{2.3}$$

Three different datasets were used. The first one corresponds to Freeman Chain Code of Eight Directions [53], that represents contours of letters from the *NIST-3* database. This codification is also used in [54, 55, 56, 44, 45]. For the Freeman chain codes, the substitution cost between symbols is equivalent to one unit for every 45 degrees of difference in the orientation of each symbol, for insertions and deletions the cost is always of two units as in [56, 44, 45] . In this dataset, we evaluated set sizes of $\{45, 90, 180, 270, 360\}$ and, for each one, 26 independent samples were drawn.

The second dataset considers 23 symbols representing different amino acids. We selected 175 samples of orthologous of insulin protein, representing 70 species, obtained from eggNog online aplication (http://eggnogdb.embl.de/#/app/home) with length ranging in [100..,300] and average 150. With them, we prepared 26 different sets in total, 5 different for each sets size of $\{20, 40, 80, 120, 160\}$ strings, respectively, and 1 set with size 175 with all the data available. We use the well-known BLOSUM62 [58] cost function.

We also generated a third dataset containing synthetic Freeman chain codes as in [56, 44, 45]. With these data, we aimed to study how algorithms scale for sets with sizes of $\{45, 90, 180, 270, 360\}$, with the average length of the strings of $\{20, 40, 80, 160, 320\}$ symbols, respectively. The length variation among strings in the same set was of 10 %. We generated 5 different sets for each possible combination of set size and string length, making a total of 125 independent sets.

We designed experiments to compare our proposal with the best algorithm described in [45], labeled as *Median-all*, and in [44], la-

beled *Frequency* and *Frequency*Cost*, in terms of edit distances calculated, average distance to median string and time.

In Fig. 2.22, we can differentiate three groups of algorithms, at the top, we see the algorithms labeled as *Frequency* and *Frequency*Cost*, exposed in [44]. These two algorithms are those with the highest number of edit distances calculated, a number that increases very rapidly as the size of the dataset grows. In the central region, we see the algorithm labeled as *Median-all*, presented in [45], and a variant that takes as a starting point the one described in section 2.3.4, labeled *HSP-all*. These two algorithms perform better compared with those mentioned above. As the dataset size grows, differences between them are better observed. Finally, in the bottom part of Fig. 2.22, the algorithms *Median-trimmed* and *HSP-trimmed* are shown. These two algorithms are the ones that perform the best.

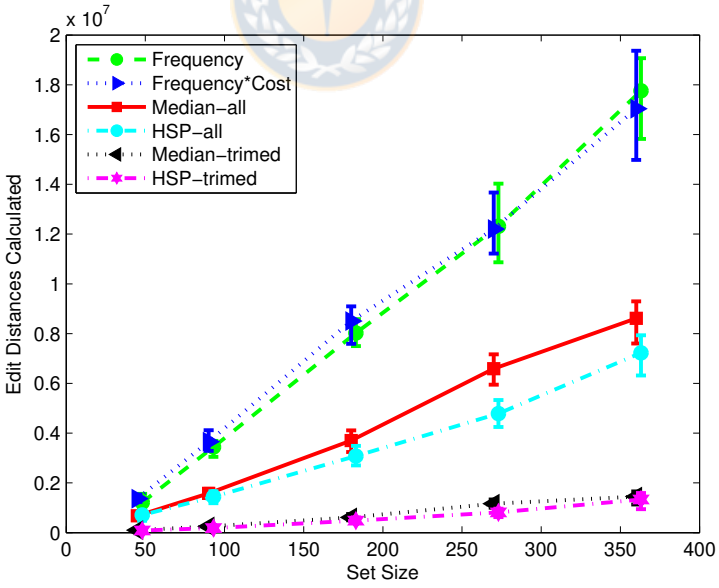

Figura 2.22: Synthetic Freeman Chain Codes. Edit Distances Calculated.

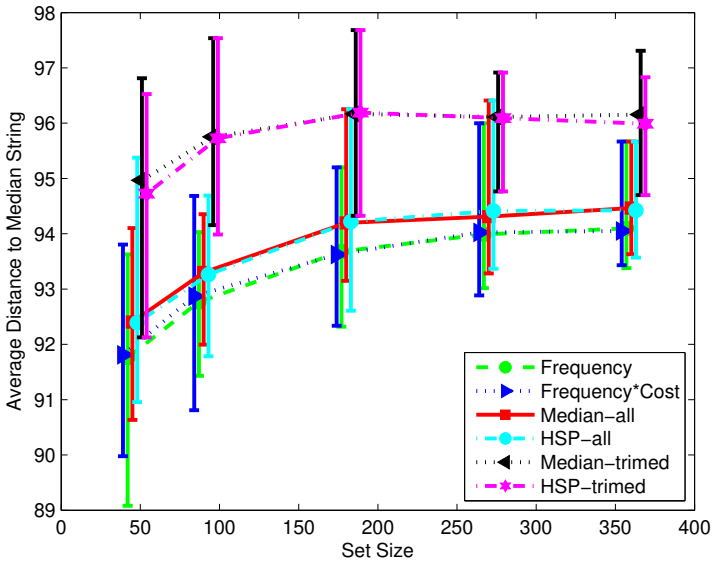

Figura 2.23: Synthetic Freeman Chain Codes. Average Distance to Median.

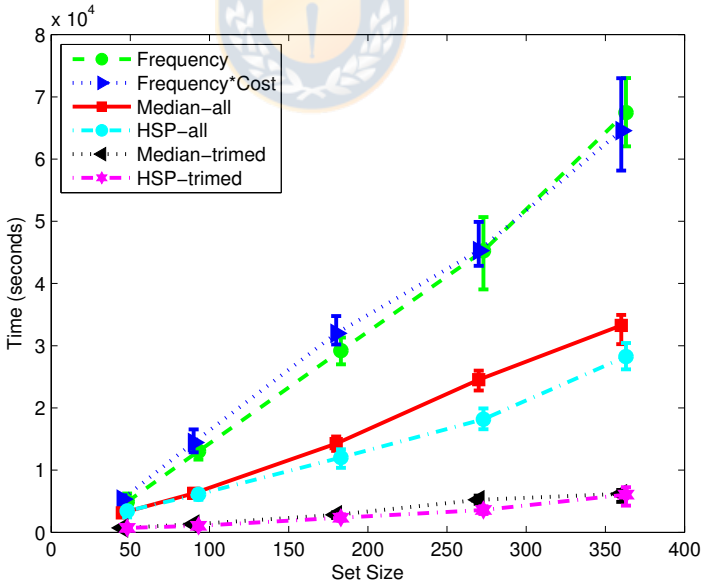

Figura 2.24: Synthetic Freeman Chain Codes. Time.

In Fig. 2.23 the quality of the solution achieved by the same algorithms is studied. As can be seen, the quality of the two algorithms computing fewer edit distances is slightly worse. In Fig. 2.23, we can differentiate two groups of algorithms, at the top, we see the algorithms labeled as *Median-trimmed* and *HSP-trimmed*. Except for these two algorithms, the others behave in a very similar way as regards to the average distance for the median string. Finally, as expected, Fig. 2.24 shows a similar behavior to Fig. 2.22, i.e. the running time of the method is proportional to the edit distances that they compute.

Next, we expose in details the effect of each modification when applied independently. In Fig. 2.25 to Fig. 2.33, we see the comparison between the algorithm that takes as a starting point the one proposed in Section 2.3.4, labeled as *HSP-all*, comparing it with the same algorithm starting from the median, labeled as *Median-all*. It is essential to clarify that we include the edit distances calculated for obtaining the starting point for those algoritms that use HSP test. For all the datasets, results show that in most of the cases our proposal requires less operations, while, as it can be seen in Fig. 2.28 and in Fig. 2.30, the quality of the median string obtained, in the Freeman Chain Codes datasets, is equivalent in both cases. Fig. 2.29 shows that, in the proteins dataset, the median string is slightly worse for *HSP-all*.

We show the comparison between algorithms with the same starting point, trimming the list of operations and without trimming it. In this analysis, two new algorithms are incorporated besides the *Median-all* and the *HSP-all*, explained in details in the previous section. We label as *Median-trimmed* the algorithm that takes as a starting point the median of the set and trims the list of operations when the expected quality of the operation is zero. We label as

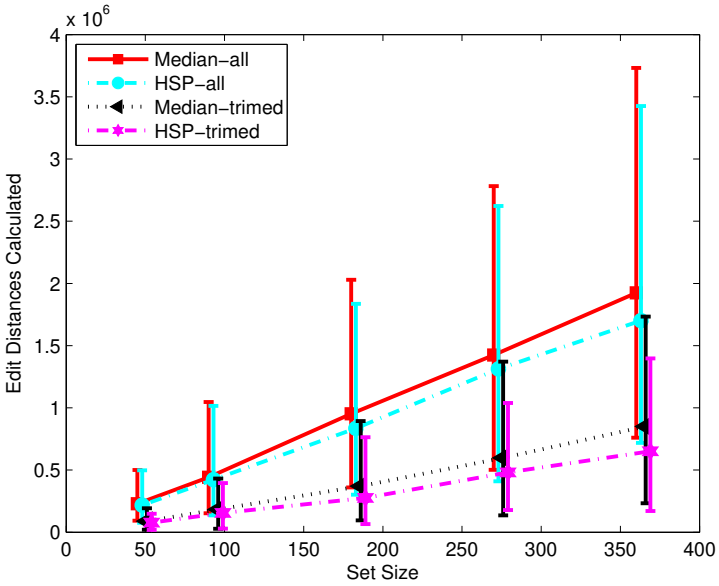

Figura 2.25: NIST Freeman Chain Codes. Edit Distances Calculated.

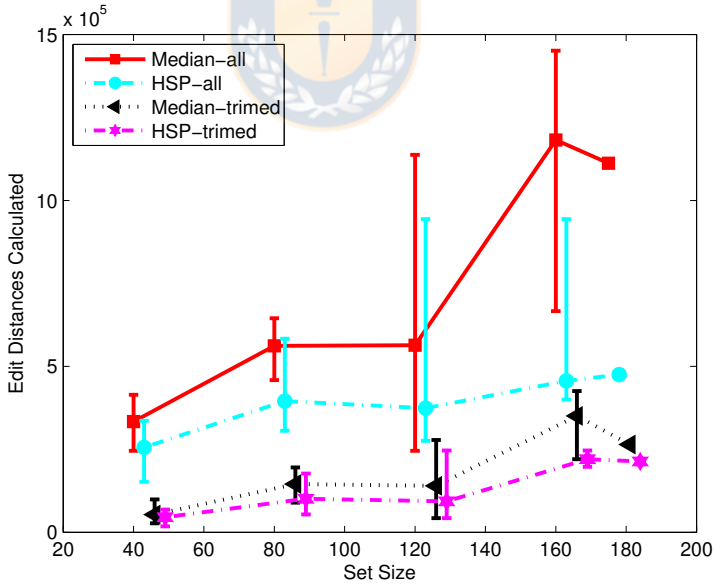

Figura 2.26: Proteins. Edit Distances Calculated.

HSP-trimmed the algorithm that takes as a starting point the one

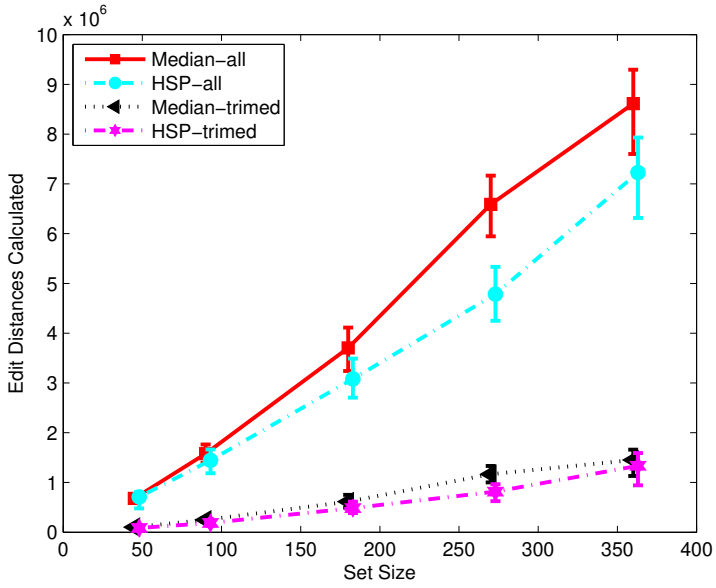

Figura 2.27: Synthetic Freeman Chain Codes. Edit Distances Calculated.

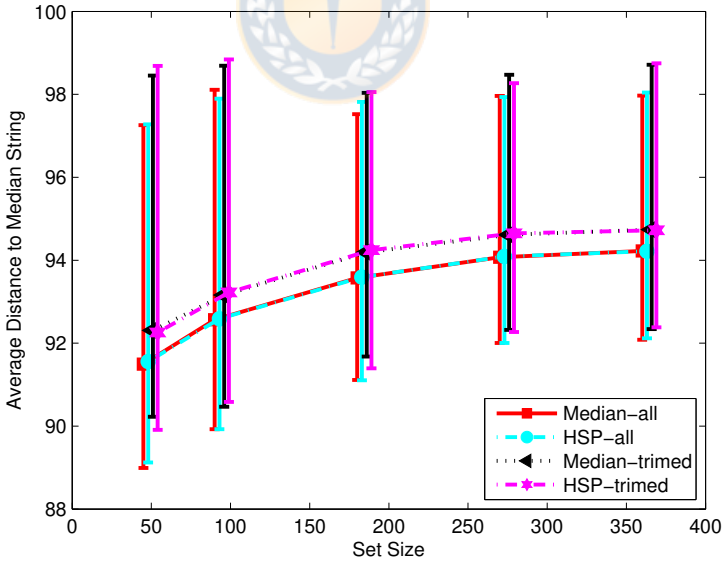

Figura 2.28: NIST Freeman Chain Codes. Average Distance to Median String.

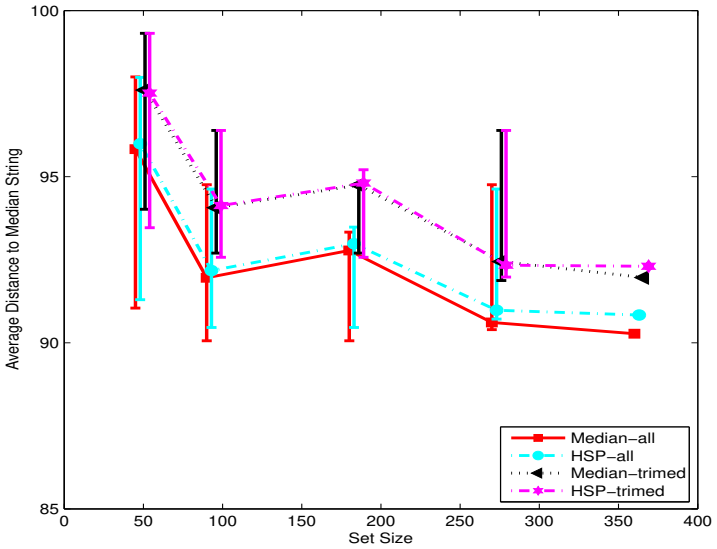

Figura 2.29: Proteins. Average Distance to Median String.

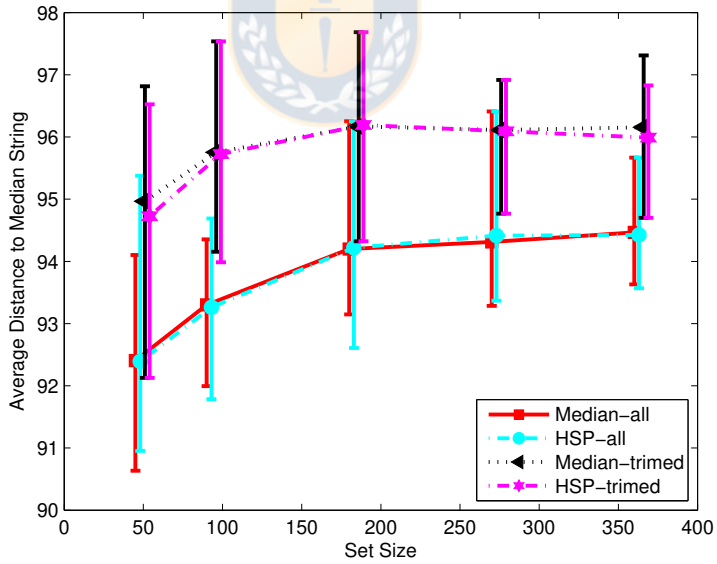

Figura 2.30: Synthetic Freeman Chain Codes. Average Distance to Median String.

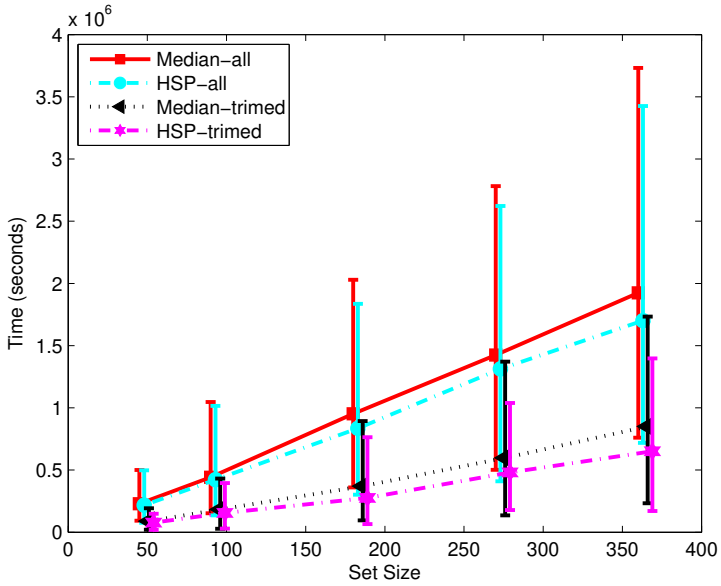

Figura 2.31: NIST Freeman Chain Codes.Time.

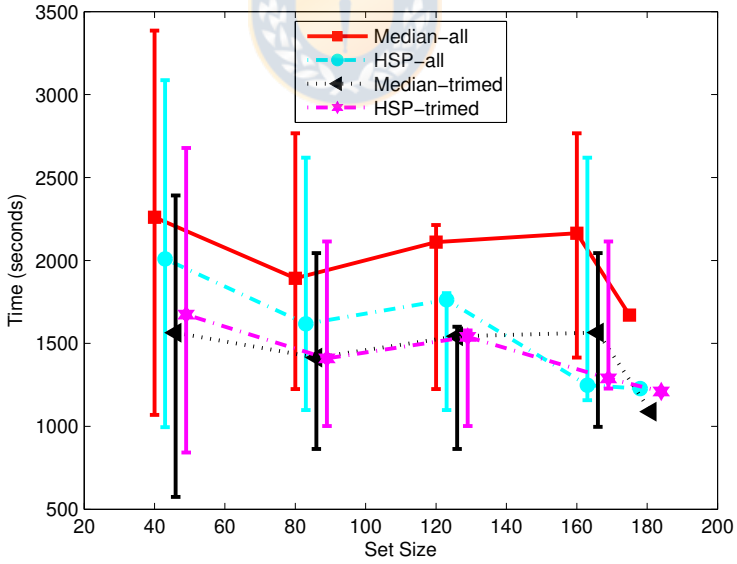

Figura 2.32: Proteins. Time.

proposed previously in Sec. 2.3.4 and trims the list of operations

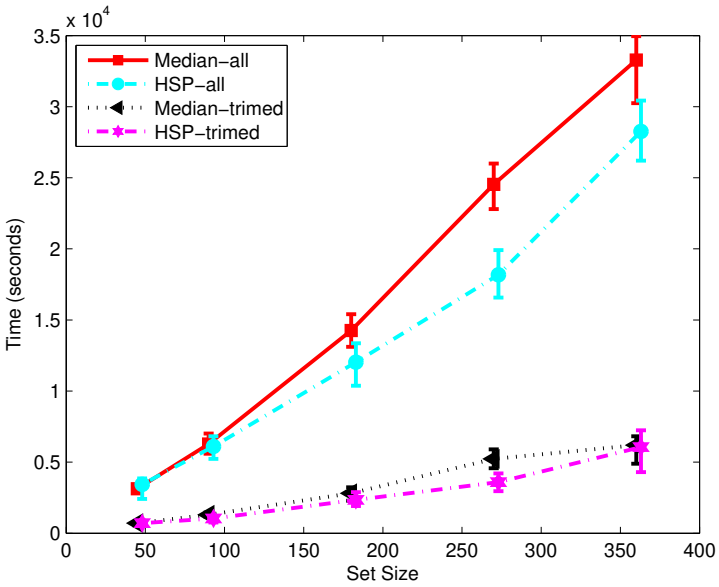

Figura 2.33: Synthetic Freeman Chain Codes. Time.

when the expected quality of the operation is zero.

In Fig. 2.25, Fig. 2.26 and Fig. 2.27 we notice that trimming can reduce the number of calculated edit distances, and this, lead to a decrease of the execution time, seen in Fig. 2.31, Fig. 2.32 and Fig. 2.33. However, the quality of the median is slightly worse when trimming, as it can be seen in Fig. 2.28, Fig. 2.29 and Fig. 2.30.

Finally, we can see the significant difference that exists, concerning the edit distances calculated and execution time, between the current state of the art, *Median-all*, and our new proposal, *HSP-trimmed*. We can also compare the quality of the median string achieved for each of the different algorithms. The results show that the loss of quality of the median string in *HSP-trimmed* is small, and can be assumed for cases in which a high speed of convergence is required.

2.3.6. Conclusions and Future Work

In this paper, a new starting point for perturbation-based iterative refinement algorithms to compute the median string is proposed with satisfactory results. This approach is based on the median of the elements of the set obtained by applying Half Space Proximal test to the median of the set. The idea of weighting elements of the set is introduced to calculate the median string in this stage, in which the weight of each element is in correspondence to the number of elements that are represented by such element. In addition, a better look at the ranking of operations allowed us to trim the list of operations when the expected quality of the operations is less than zero, which has accelerated the convergence of the algorithm. With the combination of these two results we obtain a new algorithm that improves the previous state of the art. When comparing *Median-all* with *HSP-trimmed*, on average, a reduction of 86 % of the calculated editing distances is achieved, a decrease in time of 82 % and the average distance to the median string increases in 2 %. Hence, it should be noted that this improvement in the speed of convergence of the new algorithm slightly affects the quality of the median string.

As future work, we propose to obtain a lower bound for median string, using optimization techniques such as Linear Programming and SAT Solvers. This bound will allow us to establish how close our algorithms are to the theoretical median string and establish a better benchmark. It may be interesting to study strategies that, searching only on a subset of the strings, lead to an approximation to the median string of the whole set. We can also evaluate the performance of other algorithms, presented in the related work, taking as a starting point the one obtained through the HSP strategy. Also, we will analyze these algorithms behavior when applying multiple

edits at the same time.

2.4. Median String Challenge

Título: *Median String Challenge.*

2.4.1. Abstract.

In this paper, we present a challenge for the Median String Problem. We create several datasets, divided into three benchmarks, with a variability representative of the different levels of difficulty that this problem requires. This challenge has three Benchmarked Solutions, the first one uses SAT solvers, the second one uses ILP and the third one uses Simulated Annealing. We show comparative results of the available solutions. All this work is supported by the Wayki platform, a new platform to which we encourage to send new solutions.

2.4.2. Introduction.

The problem of, giving a set of objects, find (or compute) the object that best represents all the objects in the set is a fundamental problem with applications in many domains. Actually, in such a general formulation, we are describing two different problems, one in which the *representative* object has to be part of the set, and another one in which it may be a constructed object. When the objects are strings, the *set median* is a common approach for the former, whereas the *median* string is a common approach for the later.

To motivate our work, let us present some applications in which computing the median of a set of strings may be useful. First, in the field of pattern recognition, there are applications in the recognition of handwritten characters [6]. In this example, handwritten

characters can be represented with Freeman chain codes [53], and the median of all the samples of the same character is a good representative to recognize new instances of such character.

Secondly, there are many applications in bioinformatics as nucleotide and amino acid sequences can be represented as strings. An example is the discovery of *sequence patterns that may represent important structural or functional features in nucleic acid and protein sequences*, called motifs [69]. Also, the development of Next Generation Sequencing (NGS) is drastically increasing the volume of information, hence encouraging the use of data compression techniques to store such sequences. Relative-Lempel-Ziv (RLZ) [70, 71] was proposed as a good compression scheme for this kind of data in which many similar sequences have to be compressed. In this scheme, a representative string is used as reference, and all the other sequences are compressed with respect to such reference. In [72], the authors show that the construction of a good dictionary reference may save up to 27 % space.

Relative compression has been applied in a completely different domain, which is the compression (and indexing) of trajectories of moving objects [73]. Such trajectories can be reported in different formats, being the geographic coordinates at each timestamp the classical one. When the movement of the objects is restricted to a network (for example, a road network) such trajectories can be represented as sequences, or strings, of node identifiers. Also, some work has been devoted to enrich trajectories with more application-oriented information, which has been called semantic trajectories [74]. In several applications, this semantic information can be also represented as sequences of event identifiers [75]. Hence the computation of the median string has applications in this domain, both to improve compression or to compute representative

sequences of events.

The examples above show that the median string has applications in several different domains. Not only that, but the characteristics of the sets of strings in such domains are different, both in number of elements (*size*), *length* of the strings, and *alphabet*. For example, in protein sequences a dataset may contain strings of length up to 600 built over an alphabet of size 23, whereas for Freeman chain codes the alphabet size is 8 and the strings are shorter (length up to 200). For nucleotide sequences the alphabet is even smaller (just 4 symbols). Hence, it is very important to provide a simple and objective way to evaluate and compare the performance of different methods to compute the median string, under different assumptions (i.e. datasets). This is the goal of our work, as we propose a challenge, and a platform to host such challenge, evaluating several methods of the state of the art for the median string problem. This challenge is open for the research community, so new methods can be incorporated and benchmarked.

The rest of the paper is organized as follows. First, we formalize the problem and study its complexity in Section 2.4.3. Then, Section 2.4.4 describes benchmarked solutions and other related work. Section 2.4.5 presents the challenge, including both the datasets and the platform. In Section 2.4.6 we present the evaluation results of the benchmarked solutions. We conclude in Section 2.4.7 with final remarks and future work directions.

2.4.3. Problem Definition and Complexity Issues

In this work, we study the problem of computing the *median* of a set of strings. We use the terminology in [45], in which Σ is an alp-

habet where ϵ denotes the empty symbol, Σ^* is the set of all strings over Σ, and S_i, $S_j \in \Sigma^*$. An edit operation is a pair $(a, b) \neq (\epsilon, \epsilon)$, written $a \rightarrow b$, which transforms a string S_i into S_j, if $S_i = \sigma a \tau$ and $S_j = \sigma b \tau$, where σ and τ represent substrings. Substitutions, deletions and insertions are denoted as $a \rightarrow b$, $a \rightarrow \epsilon$, and $\epsilon \rightarrow b$, respectively. The *median string*, which does not have to be unique, is a constructed string that minimizes $\sum_{S_i \in S} d(\hat{S}, S_i) | \hat{S} \in \Sigma^*$, where $d(S_i, S_j) = argmin_E\{\omega(E_{S_i}^{S_j})\}$ is the distance between both strings. $\omega(E) = \sum_{e_i \in E} \omega(e_i)$ is the cost of a sequence of operations $\{e_1, e_2, ..., e_n\}$ transforming S_i into S_j and, $\omega(a \rightarrow b)$ is a domain-specific function that assigns a cost to each edit operation. This problem definition is due to [5].

The definition above includes an edit distance that can be defined in different ways, being Levenshtein distance [26] probably the most studied one. In this distance definition, each edit operation has unitary cost (i.e. $\omega(a \rightarrow b) = 1$).

The complexity of the problem has been studied. First, [34] showed that, if Σ is unbounded, computing the median string corresponds to a NP-complete decision problem. Later, in [32], it is shown that the problem is NP-complete when the penalty matrix satisfies the triangle inequality (i.e. it is a metric). Finally, [33] showed that the problem is NP-complete under the Levensthein distance, even if Σ is binary. Therefore, different approximations have been proposed to compute the median string [41, 5, 14, 42, 43, 45].

As mentioned in Section 2.4.2, this problem is related with the *set median* (i.e. a string in the set that minimizes $\sum_{S_i \in S} d(\hat{S}, S_i) | \hat{S} \in S$). The set median is sometimes used as an approximation to the median. Also, several methods to compute the median string use the set median as an initial candidate solution.

2.4.4. Related Work and Benchmarked Solutions

An exact algorithm to compute the median string was first described in [17]. For the Levenshtein [26] metric, it requires a computational time of order $\mathcal{O}(l^N)$, for a set S of N strings of length l. Due to the exponential dependence on N, this approach may not scale well as the number of strings increases. For that reason, several heuristics has been described.

Many proposals belong to the so called perturbation-based heuristics [76]. This approach attempts to increase the quality of the approximated median $\hat{S}$, i.e. minimizing $\sum_{S_i \in S} d(\hat{S}, S_i)$, through successive refinements of an initial string such as the set median, the empty string or a greedy approximation.

Some authors [5, 31, 48, 42] studied blind search strategies, in which perturbations are tested in a predefined order without an estimation of its goodness that can guide the search.

In [31], the approximated median is assembled symbol by symbol. In each iteration a goodness function is applied to select a symbol that is appended to $\hat{S}$, which initially is the empty string. The tie-breaking criterion described in [47] can be applied when more than one symbol have the same goodness.

Another alternative described in [5], is to start from the set median adjusting the current solution by applying insertions, deletions and substitutions at every position, in order and one at a time. If the edit leads to an improvement, it is accepted. It is worth to note that the i-th perturbation is performed not over the initial string but over the string that results from previous editions if any was accepted.

Different to [5], in [48, 42] at the i-th iteration all possible operations are applied to the i-th symbol of the initial string. The initial string for the next iteration is selected as the one minimizing

$\sum_{S_i \in S} d(\hat{S}, S_i)$ from the pool of newly generated strings together with the seed in the current stage. Also, it was proposed a variant that splits each string in d sub strings, from where result the sets $\{S_1^1 ... S_n^1\}$, $\{S_1^d ... S_n^d\}$ where S_i^j is the i-th substring of the i-th string in S. Then, the median of each set is computed, and the final median is the concatenation of the former ones in order.

Goodness functions to rank possible perturbations and greed procedures have been proposed in other works [14, 48, 43, 44, 45]. The use of such functions speed up the heuristics while achieving medians of good quality.

To further speed up the computation, in [48] a local optimization technique was also studied. Not all symbols are tested in each position, only those that fulfill a heuristic criterion, for example, the two symbols with lower substitution cost relative to the current symbol at position i.

The idea of ranking operations and the application of multiple perturbations at once was also studied by [14] and [43]. The distances from the current approximation $\hat{S}$ and each string in S are computed, applying to $\hat{S}$ the most frequent edit operation in each position.

When edit operations have cost other than one, an expensive but no so frequent operation may be better than a cheaper and frequent one. This problem, as well as whether to apply multiple or one perturbation at a time, were addressed in [44]. A goodness function based in [6] estimated how much a single perturbation will lower $\sum_{S_i \in S} d(\hat{S}, S_i)$. This approach converges to an approximated median of the same quality as in [48, 42], but it is about 8 times faster. Experiments suggest that methods applying parallel perturbations are near to 30 times faster but converge to worst solutions.

Being $E = \{e_1, e_2, ..., e_n\}$ the sequence of edit operations trans-

forming $\hat{S}$ into a string S_i, from results in [6, 44] follows that, if some of the e_i is applied to $\hat{S}$ to obtain a new $\hat{S}'$, then $d(\hat{S}', S_i) \leq d(\hat{S}, S_i)$. The case when this holds also for edit operations $e_i^{'} \notin E$ is examined in [45], which proposed a better estimate of whether a perturbation will lead to an improvement. A through empirical evaluation shows that this approach is faster than [48, 42, 44] while converging to same quality approximations to the median string.

An iterative approach is proposed in [51]. Set S is partitioned in pairs, $< s_i, s_j >$ looking to maximize $\sum_{i,j} d(s_i, s_j)$ since this is a lower bound for the sum of distances from the true median [77]. After that, the weighted median [6] of each pair is computed. Using a fitness function, the best of these medians is selected to replace the worst strings in S. If the lower bound is not reached, the procedure is repeated using the new S.

A different approach to perturbation based methods is to transform or embed the strings into a new space, computing the median in such space, and going back to the string space afterwards. This idea was explored in [36]. Harnessing that attributed strings represent shapes encoding the angle and length of segments, each string is transformed into a piecewise linear function. The median of $|S|$ strings is computed as the median of the same number of piecewise functions. The embedding idea is studied in [37], a string s is represented by a vector in $\mathbb{R}^m$, where each component is the distance from s to a prototype p_i. The median of S is computed as the median of the corresponding vectors and transformed back into a string using the results in [6] and the edit path between two strings.

The online version of the median string problem arises when strings in S are not available all at the same time, but arrives at a different time-lapse. The idea is to avoid computing the median from scratch upon the arrival of new strings. As new strings are

available, [14] computes the distance to $\hat{S}$, increasing counters for different possible edit operations. When a counter reaches a given limit, the operation is applied to $\hat{S}$. In [78], the new median is computed as a weighted mean from the former and the string that comes in. $\hat{S}$ is edited with a subset of the edit operations resulting from the distance from it to the new string.

A generalization of the median string problem for the stochastic edit distance and multi-strings is covered in [49]. In multi-strings, each position encodes a stochastic vector with each component representing the probability of a given symbol in such position. The stochastic edit distance measures the probability that a string s_i becomes s_j given probabilities for the possible edit operations. In this setup, the median is modeled as a multi-string of fixed length and the estimation of the probabilities as a maximum likelihood problem.

Finally, some authors [38, 39, 40] have studied linear programming models for the median string problem. In [38], the model aims to minimize the sum of the distance from the median to each string. Constraints related to the triangular inequality involving every pair of strings as well as the median and the non-negativity of the distance are included. This work proposes a lower bound for $\sum_{S_i \in S} d(\hat{S}, S_i)$ and analyses cases where this value can not be achieved. However, the proposed model does not allow the computation of the median string.

Since it is very important to our proposal, we included in table 2.13 a summary of the datasets used in previous works. Parameters such as alphabet size, set size and string length were reviewed. We aggregated different works attending to the type of string, for example, Freeman chain codes, natural language, RNA, among others. We can corroborate the heterogeneous experimental setup of

Cuadro 2.13: Summary of datasets utilized in previous work. For garbled text, german, english and spanish vocabularies has been used.

Datatype	Alph. size	Set size	Str. length	Works
garbled text	up to 69	5..13560	up to 448	[31, 47, 79] [38, 78, 37]
attributed str.	infty	2..98	-	[6, 78]
chain codes	8	5..360	up to 264	[79, 44, 45]
chromosomes	11	20	-	[48, 42, 44]
synthetic	up to 70	up to 3000	up to 320	[14, 44, 45]
protein seq.	23	up to 720	100..600	[45]
RNA/DNA	4	2..10	2..10	[39, 40]

works addressing the median string problem.

Benchmarked Solutions

An Integer Linear Programming model capable of computing the exact median string is described in [39, 40]. The edit distance is modeled as a path-finding problem in the correspondent weighted edit graph [29] and there is a graph for each string in S. In this setup computing the edit distance means to select edges that form the shortest path, being the cost of this path the value of the distance. The model includes variables encoding the symbols that form the median string and whether an edge is selected or not. Also, some constraints ensure that the selected edges assemble a valid path. The model minimizes the sum of the cost of the edges included in an optimal path from each graph. As in [39] we call this approach ILPMed and it will be one of the benchmarked solutions in this work. To compute the exact median, another variant that includes constraints related to the triangle inequality is also described. Besides, a model that can compute an approximated median is also covered.

We also considered a SAT encoding of the problem that translates the ILP model, described above, into SAT. The encoding of the model is straightforward. Each constraint in the ILP model is translated into a propositional clause. The only important difference between the SAT model and the ILP one is that SAT does not need to use the Big-M encoding for implication constraints, since those kind of constraints are native in the SAT language.

The other solution that we benchmarked uses Simulated Annealing to select the perturbation that will be applied to the approximated median $\hat{S}$, we call it SANMed. As in [45] we compute the edit distance from $\hat{S}$ to each string in S, and from the edit sequences, we get the operations related to each position of $\hat{S}$. In each iteration, one operation is randomly selected from this list and applied to $\hat{S}$. If this perturbation leads to an improvement, a new iteration begins, if not, there is a probability that the operation is accepted anyway. This probability is proportional to a temperature parameter that decreases as the algorithm goes on.

2.4.5. The Challenge

Challenge Platform

Wayki (`www.wayki.co`) is a project developed by GIOCo (The Research Group on Combinatorial Optimization at Universidad de Concepción). Wayki is a Web-based Platform to Support Research on Hard Optimization Problems. It is currently running a Beta release, since September, 2019. At its current stage, Wayki's main functionalities are:

- Organizers can setup a new space for a new optimization problem. For this, a "problem manager" needs to set: a problem

name and description, personalized banner, input-output formats, solution-checking script and initial benchmarks.

- Registered users can upload new results for a specific problem on the publicly available benchmarks.

- Users can access and download a challenges information (i.e. benchmarks, performance data and solutions). So far, only any-time execution logs and solutions of researchers' solvers are stored. Hence, the hardware is not uniform.

The Median String Problem has been proposed as a challenge in `http://www.wayki.co/challenge/medianString` and initial solutions for the datasets presented above can be consulted.

One of the main characteristics of Wayki is that it encourages to pay attention to the any-time behavior of the solving approaches. When trying to solve hard optimization problems that may appear in several applications, it is usually difficult to establish one solving technique above the rest in all scenarios. For instance, for different time limits, different solving techniques may be the one prevailing. Wayki encourages researchers to submit several solutions for the same benchmark file, while also providing the time at which such solutions were found. This allows the user of the platform to analyze the any-time behavior of the solving algorithms. At its current stage, Wayki's information relies on the data provided by the researchers and does not guarantee that all experiments are performed in the same hardware infrastructure. Hence, right now, Wayki presents the problems as Technological Challenges, were no constraints are considered over the resources that were needed to arrive to the submitted results.

Problem Data

As we stated in Section 2.4.2, the median string problem has applications in many different domains, in which the characteristics of the strings may be quite different. Hence, for our challenge we generated several synthetic datasets that represent this variety. The three main variables that characterize a domain are the number of strings N, the length of each string l, and the size of the alphabet σ. In addition, generated datasets are grouped into three benchmarks, each of them with an underlying motivation. The base name for each dataset is bench$X_N_\sigma_l$, where X is the identifier of the benchmark. Some of the benchmarks may have additional parameters, that are explained below.

The datasets also include a table that defines the cost of each possible operation. This table is of size $(\sigma+1)\times(\sigma+1)$ and for each cell (a, b) with $a, b \in \Sigma$, it defines the cost of substituting symbol a for symbol b. The first row and column, represent the empty string. Hence, the cost of inserting each symbol is stored in the first column, whereas the cost of deleting each symbols is stored in the first row. The main diagonal is filled with zeroes, as it corresponds to the substitution of a symbol by itself.

Benchmark 1 is the most common simplification of the problem. All the datasets in this benchmark define a cost of 1 for all the operations (insertions, deletions and substitutions). In addition, all the strings in each datasets are of the same length. The benchmark includes datasets with $N \in [8, 1024]$ to study the scalability of the methods on the number of strings. Similarly, $\sigma \in \{2, 4, 8, 23\}$ allows the study of the influence of the alphabet size. These values are inspired by the domains of application of the median string. A $\sigma = 2$ represents an ubiquitous binary alphabet, $\sigma = 4$ for amino acid sequences, $\sigma = 8$ is used in Freeman chain codes, and $\sigma = 23$

for protein sequences. Finally, the length of the strings ranges from 8 to 1024.

Benchmark 2 also contains datasets in which all the strings are of the same length, but without the restriction of all the operations having unitary cost. Hence, the name of each dataset includes an additional parameter that indicates the maximum cost MAX of each operation (i.e. each operation has a random cost in the range $[0, MAX]$). The benchmark includes datasets with $MAX \in \{1, 4, 16\}$, where value 1 is a generalization of benchmark 1 in which each cost may be zero or one, 4 is inspired by Freeman chain codes, and 16 is related with Blosum, a substitution matrix used for sequence alignment of proteins. For all the other parameters, N, σ and l, we keep the same values and motivation of benchmark 1.

Benchmark 3 is equivalent to benchmark 1 regarding N, σ and l, but the strings in each dataset are not randomly generated. In each dataset of this benchmark, an initial string is randomly generated, but all the other strings are built as mutations of such string. Hence, the datasets in this benchmark include an additional parameter $m \in [0, 100]$ that represents the probability of mutation of each symbol. An $m = 0$ generates strings that are all identical to the seed, whereas $m = 100$ generates completely independent strings (making such configuration equivalent to benchmark 1). The motivation for this benchmark is that, in many domains, just makes sense to compute a representative of the set if there are some similarities between the elements of such set. For example, for handwritten characters, it is common to compute the median of all the instances of the same character, but not so much to compute the median of instances of different characters. In other domains, such as amino acid sequences of individuals of the same species,

strings are intrinsically similar.

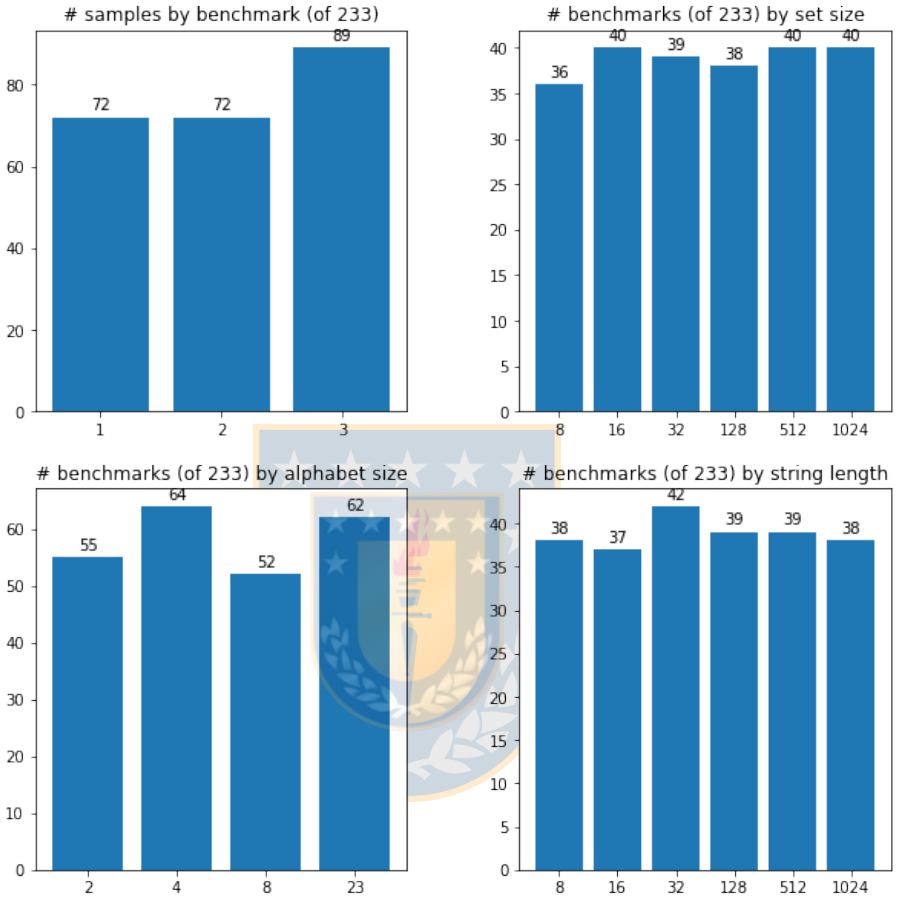

Figura 2.34: Benchmark Features

In Fig. 2.34, we show some features of the datasets. As shown, in Fig. 2.34 top left, we have 233 datasets, 72 in the 1 and 2 benchmarks, and 89 in the 3 benchmark. In Fig. 2.34 top right, we create between 36 and 40 datasets for each set size selected. In Fig. 2.34 boton left, we show that we have between 52 and 62 datasets for the selected size of alphabet, and Fig. 2.34 botom right, we see that we have between 37 and 42 datasets for each string length selected.

2.4.6. Results and Discussion

After performing the experiments, we found that not all algorithms managed to report a solution within the time limit established for this challenge. We also found that both SAT and ILP, have greater limitations than SA, in terms of the maximum size of the problems solved. If we order the algorithms, based on the size of the problem they can handle, we have SAT as the most limited, then, and without major differences we have ILP, and as the one that can find solutions for bigger problems we have SA.

In Fig. 2.35 we see that SA achieves solutions for 80 % of cases in datasets with size 8, 16, 32 and 128, decreasing to 65 % in dataset with size 512 and to 45 % in in dataset with size 1024. In the same figure, we observe that SAT only reports solutions for 5 % of datasets with size 1024 and about 20 % in datasets with size 512, while for the rest of dataset sizes vary between 25 % and 30 %. This is very similar to what happens with ILP, that report solutions only for 15 % of datasets with size 1024 and 512, while for the rest of dataset vary between 25 % and 35 %.

Regarding the size of the alphabet, in Fig. 2.36 we see that SA achieves solutions between 55 % and 80 % of datasets for alphabets with size 2, 4, 8 and 23, on the other hand, both SAT and ILP report solutions for less than 20 % in datasets with size 4, 8 and 23, only exceeding this percentage in datasets with binary alphabet, where they achieve solutions in 30 % of datasets.

Analyzing the behavior of the different algorithms regarding the length of the strings, we find the following results that we present in Fig. 2.37. SAT, could not handle instances of the problem for strings lengths greater than 32, for strings with length 8, reports solutions for 55 % of the datasets, a value that is decreasing as the size of the strings increases, reporting solutions for 55 % of datasets

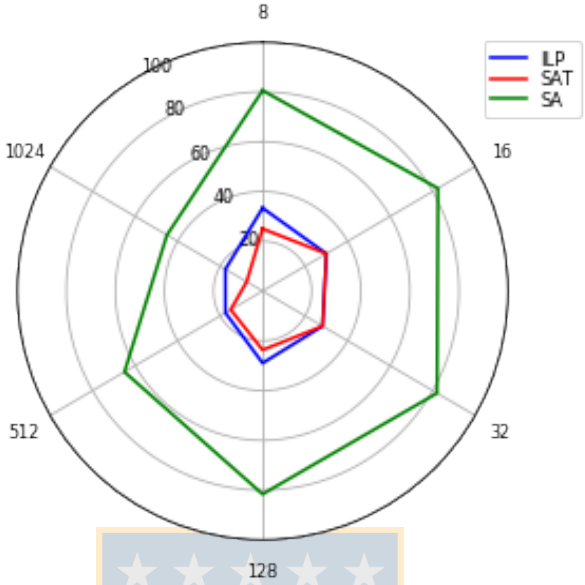

Figura 2.35: Percent of benchmarks with solutions vs set size

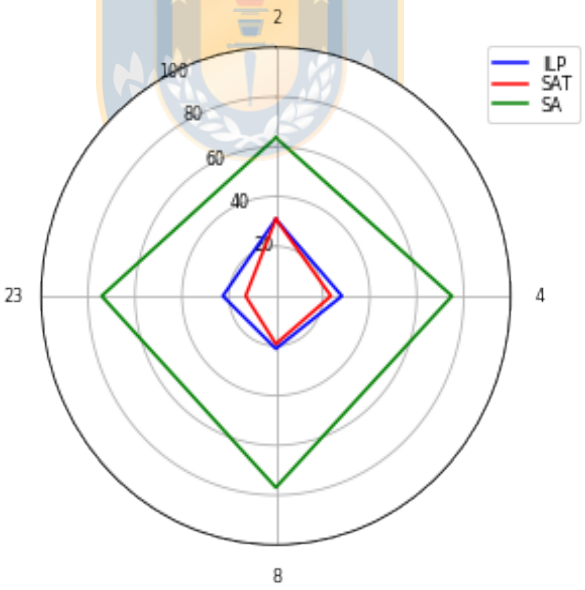

Figura 2.36: Percent of benchmarks with solutions vs alphabet size

with strings with length of 16 and 30 % for datasets with strings with length of 32. In the case of ILP, the situation is slightly better,

because although it does not report solutions for strings with length of 1024 and 512, it achieves solutions for 5 % of datasets with strings with length of 128 and between 60 % and 40 % for strings with length of 8, 16 and 32. In the case of SA, it is by far the one that finds solutions for the greatest amount of datasets, being in all cases superior to 60 %, reaching 90 % in datasets with string length of 128.

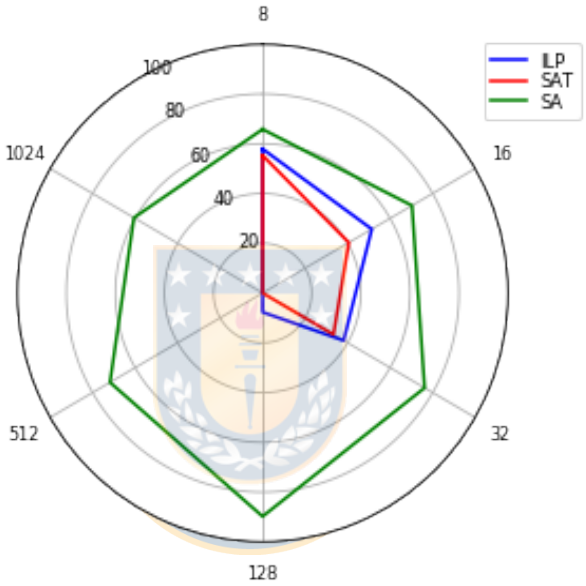

Figura 2.37: Percent of benchmarks with solutions vs string length

Another result that we want to expose, is the sensitivity of these algorithms for variations of any of the parameters taken into account in these datasets. Due to the few cases of datasets for which ILP and SAT have reported solutions, the analysis is done using the behavior over time for the SA algorithm. In Fig. 2.38 we see how the time increases with respect to variations in the number of strings in the dataset, in the case of Fig. 2.39 we can see this effect caused by the size of the alphabet and in Fig.2.40 we see that it is caused by the length of the strings in the datasets. In the three cases

mentioned above, the only parameter that varies is the one to be analyzed. Simultaneous variations of these parameters would show time behavior in orders of magnitude different for the datasets. On the quality of the reported solutions, optimal solutions were found for 4 benchmarks in which the binary alphabet was used, in all these cases SA found the optimum.

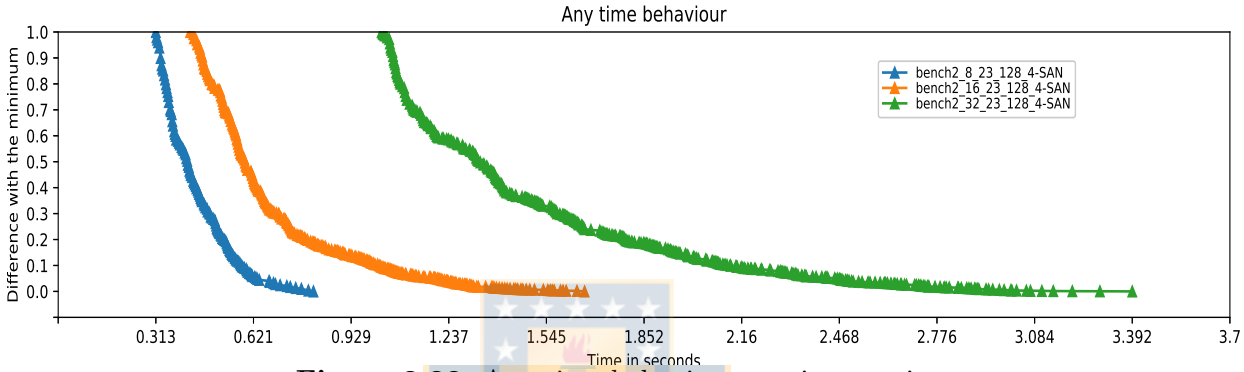

Figura 2.38: Any time behaviour varying set size

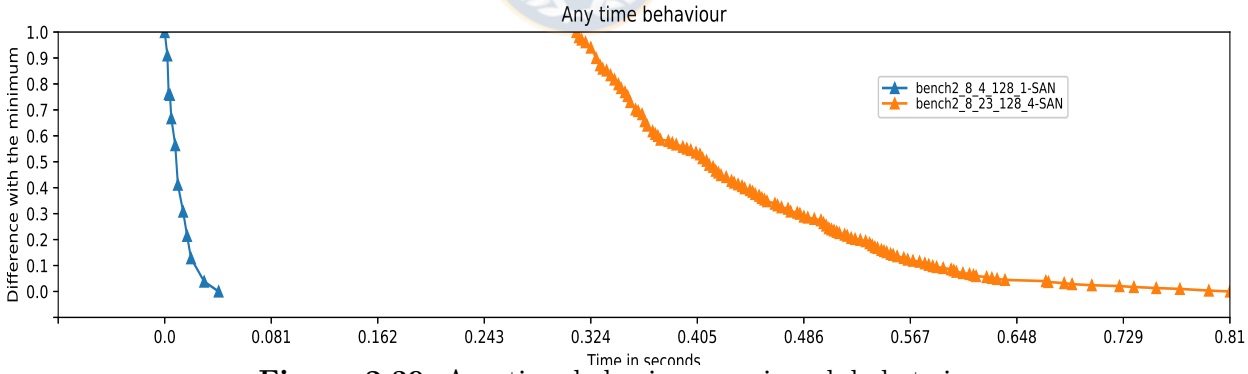

Figura 2.39: Any time behaviour varying alphabet size

2.4.7. Conclusions and Future Work

In this paper we present a challenge for the median string problem, for that, we design benchmarks with great variability in each

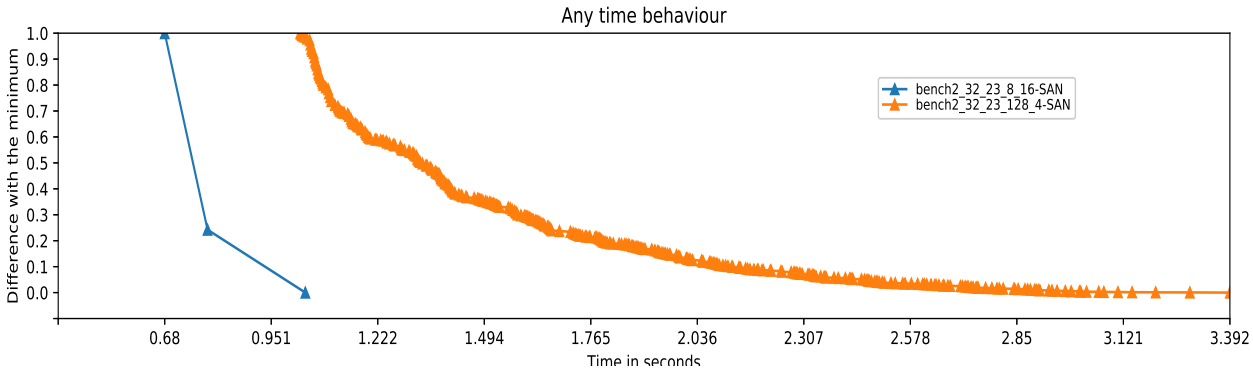

Figura 2.40: Any time behaviour varying string length

of its parameters, so that the algorithms that are tested can be sufficiently validated. Among the algorithms that have been tested, SA achieves better results in terms of the number of datasets in which it finds solutions and the time it finds them. The quality of the solutions provided by SA has been validated in all cases in which ILP or SAT have found an optimal result, which does not necessarily mean that in all cases where a result is found, it is close to the optimum. As future work, we will work to include within this challenge the results of other algorithms present in the related works and to encourage the community to send their results. Also, we plan to incorporate other challenges to the Wayki platform, in order to improve the platform and have more feedback on its operation.

Capítulo 3

Discusión, conclusiones y trabajo futuro

En este capítulo se presenta un compendio de los principales aportes hechos a los algoritmos para el cálculo de la cadena media, así como los resultados que los avalan y las ideas que, aun siendo importantes, no han podido ser exploradas durante esta tesis y que constituyen líneas interesantes de trabajo futuro. En la Sección 3.1 se hace un análisis de los resultados obtenidos, vistos con un enfoque transversal a los artículos científicos expuestos en el Capítulo 2 y se exponen las conclusiones a las que se ha podido llegar. Finalmente en la Sección 3.2 se exponen las líneas de trabajo que han quedado abiertas con el desarrollo de esta tesis.

3.1. Discusión y conclusiones

En el proceso de validación de las hipótesis planteadas, el cumplimiento de los objetivos específicos y el desarrollo de las tareas planificadas, se han puesto de manifiesto resultados que permiten discutir los siguientes que se enuncian a continuación.

La propuesta de modificación del ranking de posibles ediciones que se ha presentado, en el que se considera la repercusión de cada una de las posibles ediciones sobre todas las cadenas del conjunto, es una método más efectivo comparado con el que se empleaba anteriormente y en el que solamente se consideraba el efecto de las ediciones en las cadenas del conjunto en cuya secuencia de transformación óptima se encontraba presente la edición analizada. Esta afirmación está avalada por los experimentos expuestos en la Sección 2.1.6 en los que se muestra cómo esta variante del algoritmo mejora en términos de velocidad de convergencia a los algoritmos considerados estado del arte al momento de ese trabajo, presentados en [44], conservando la calidad de la solución obtenida. Esto valida la primera hipótesis planteada en esta tesis.

En los algoritmos presentados en [44], el rango de valores que podían ser asignados a una edición, correspondía a valores mayores o iguales que cero, donde cero es equivalente a no conocer o considerar nulo el efecto de esa posible edición en la reducción de la distancia a la cadena media y los valores mayores a cero correspondían a la reducción estimada de la distancia entre todos los elementos del conjunto respecto a la cadena media. A partir del nuevo ranking, el rango de valores pasa a contener también valores negativos, asignados a las ediciones que se estima que pueden empeorar la distancia a la cadena media respecto a todas las cadenas del conjunto. Con esto, se abre la posibilidad de concentrar mejor el esfuerzo del algoritmo en las cadenas más prometedoras, llegando incluso a desechar operaciones con valores de ranking negativo, idea que es explotada en el artículo expuesto en la Sección 2.3.

Si bien los resultados del primer artículo presentado son prometedores, se decidió profundizar en la idea de continuar ganando en velocidad de convergencia sin afectar a la calidad de la aproxi-

mación de la media encontrada. Para esto, se buscaba trabajar con un subconjunto de las cadenas. En este reto destaca la incorporación a los algoritmos de la capacidad de trabajar ponderando las cadenas a las que se quiere calcular la media, donde se hacía una equivalencia entre el peso de cada cadena y la cantidad de cadenas representadas por ella. En la Sección 2.2.6 se logra establecer que para los conjuntos de datos estudiados se puede reducir como promedio en torno a un 8 % la cantidad de distancias de edición que es necesario calcular, los mejores resultados son alcanzados cuando se trabaja con un subconjunto de elementos que ronda el 90 % de elementos sin afectar la calidad de la aproximación de la media encontrada. A pesar de lo anterior, y teniendo en cuenta que en algunos de los *datasets* no se logra mejoría, se puede considerar que este enfoque, aunque es tentativo, no conduce inequívocamente a la mejora de los algoritmos, por lo que podría ser aplicado de forma discrecional. Esto valida la segunda hipótesis planteada en esta tesis.

Otro resultado importante obtenido en esta tesis se deriva de los resultados del artículo presentado en la Sección 2.3, en el que se muestra cómo se puede utilizar, como punto de partida, uno diferente a la cadena perteneciente al conjunto que menos distancia acumule al resto, siendo ese el enfoque predominante en el estado del arte. La elección propuesta como punto de partida, es obtenida a partir de los vecinos del candidato anteriormente mencionado, utilizando para detectarlos el Algoritmo 5, descrito originalmente en [46]. A estos vecinos se les calcula la media y ésta es utilizada como punto de partida.

El Algoritmo arriba mencionado, es un test local, que se aplica independientemente a los vértices de un grafo, devolviendo un subconjunto del grafo, estrechamente relacionado con el vértice al que

se le realiza el test. Un requisito para aplicar este test es que tiene que ser utilizado sobre espacios métricos, lo que hace posible su uso en este dominio.

Además de utilizar una alternativa al punto de partida del algoritmo presentado en la Sección 2.1, se saca provecho, como se explica al inicio de esta sección, del nuevo ranking de evaluación de las posibles ediciones, estableciendo como criterio, el no considerar ediciones que según ese ranking pronostiquen un aumento de la distancia a la cadena media. Cada una de estas modificaciones por separado logra disminuir tanto el tiempo de cómputo como la cantidad de distancias de edición que es necesario calcular. Los efectos más notorios se encuentran al combinar ambas modificaciones, llegando a lograrse una reducción del 86 % de la cantidad de distancias de edición que es necesario calcular y del 82 % del tiempo de cómputo. Sin embargo, cabe mencionar que esta mejora afecta ligeramente la distancia a la cadena media respecto al Algoritmo 1, aumentando un 2 %. Por tanto, esta aproximación es de aplicabilidad en dominios donde se necesita rápida convergencia sin importar sacrificar levemente la calidad de la solución obtenida. Esto valida la tercera hipótesis planteada en esta tesis.

Como cierre, en la Sección 2.4 se evalúa la calidad de la solución encontrada por nuestro algoritmo comparándola con el óptimo real. En dicha sección se expone un *Challenge* para el cálculo de la cadena media en el que los métodos seleccionados, para validar la propuesta aquí presentada, son los modelos que usan SAT e ILP, pues se ha podido hacer una comparación en los casos que se obtiene el óptimo. Es de resaltar que el representante de la familia de los algoritmos basados en perturbaciones iterativas aquí presentado SA encuentran una solución óptma para todos los casos en los que o SAT o ILP también lo hacen. Cabe señalar, que el hecho de

que los modelos que utilizan SAT e ILP solamente puedan manejar instancias pequeñas del problema, no garantiza que los algoritmos propuestos en esta tesis mantengan el mismo nivel de optimalidad como con instancias pequeñas, de todas formas los resultados obtenidos, a pesar de no ser concluyentes, son alentadores.

Por tanto, a modo de conclusión se ha logrado cumplir con los cuatro objetivos específicos propuestos en la Sección 1.4, en particular, para el objetivo G1 se ha propuesto el Algoritmo 1, que proporciona un mejor ranking de cada una de las operaciones de edición, logrando estimar de mejor manera el efecto de una perturbación en la expresión que defina la cadena media. En cuanto al objetivo G2, mediante los algoritmos 2, 3 y 6, se logra reducir como promedio las operaciones de edición en un 8 % en los datasets estudiados, resultado logrado al aplicar el enfoque de Selección de Pivotes sobre Espacios Métricos al problema de la cadena media. En cuanto al objetivo G3, logramos encontrar que, mediante el Algoritmo 5, era posible identificar un conjunto de cadenas estrechamente relacionadas con la mediana del conjunto, a partir de las cuales, calculamos un nuevo punto de partida para el algoritmo de la cadena media, teniendo mejores resultados como punto de partida si se compara con utilizar la mediana del conjunto. Un resultado adicional, que no tributaba a ninguno de los objetivos específicos, fue la validación de un valor de poda para las posibles ediciones, que permitía evitar evaluar aquellas operaciones que empeoran la solución parcial encontrada. Por último, en relación al objetivo G4, se muestra que el error de los algoritmos diseñados se encuentra bien acotado, ya que en los casos en los que se pudo obtener una solución óptima mediante SAT o ILP, se llegó al mismo resultado mediante un algoritmo basado en perturbaciones iterativas como los presentados en esta tesis.

3.2. Trabajo Futuro

Si bien se ha mostrado durante el desarrollo de esta tesis avances significativos en los algoritmos para el cálculo de la cadena media basados en perturbaciones, todavía quedan abiertos temas relacionados a este problema que podrían ser abordados en el futuro.

Dentro de las líneas que pudieran dar continuidad a este trabajo, se pueden señalar varios grupos, donde el primero, englobaría a los problemas similares al tratado en esta tesis en los que se pueda extender los resultados alcanzados. Propuestas como extender los algoritmos diseñados al cálculo de la cadena central [80], estarían dentro de las derivaciones directas de este trabajo. El problema de la cadena central es similar al de la cadena media, con la diferencia que en vez de buscar la cadena que menos distancia acumule al resto, busca alternativamente la cadena que minimice la distancia máxima a cada una de las cadenas del conjunto. Otra de las extensiones naturales que se deben explorar es adecuar los algoritmos creados para que puedan emplearse bajo distancias distintas a la distancia de edición de Levenshtein, como principal candidata a explorar podría encontrarse la distancia de Damerau-Levenshtein [27], muy similar a la distancia de Levenshtein, pero que además considera como válidas las operaciones de intercambio de símbolos.

Quedan también por explorar, alternativas de algoritmos iterativos basados en perturbaciones, que intenten aplicar múltiples operaciones en cada iteración, similar a lo intentado en [14, 43], pero utilizando en nuevo ranking expuesto en esta tesis. También explorar alternativas de algoritmos iterativos basados en perturbaciones que no tengan un comportamiento voraz, como podría ser el caso del enfriamiento simulado [81] o variantes de algoritmos evolutivos como los algoritmos genéticos [82]. El correcto ajuste de los

parámetros necesarios para ejecutar cada una de las dos variantes mencionadas anteriormente pudieran requerir un arduo trabajo, pero es una tarea que no se debe descartar. De hecho, en [6] se presenta una aproximación a este problema utilizando algoritmos genéticos, por lo que resultaría interesante tomar sus resultados como marco comparativo.

Por otra parte, se debe plantear la posibilidad de implementaciones para estos algoritmos que sean capaces de aprovechar las facilidades en términos de paralelismo que pueden ser explotadas. Identificamos como paralelizables varias etapas presentes en el algoritmo 6. Una de estas etapas es cuando se calcula en cada iteración la calidad de las cadenas candidatas, donde se necesita calcular la distancia entre la candidata y todas las cadenas del conjunto, siendo estos cálculos independientes. La otra etapa del algoritmo que es potencialmente paralelizable es el de la estimación de la calidad de las operaciones de edición, ya que este análisis se hace independientemente en cada una de las posiciones de la cadena.

Por último, es interesante aplicar estos algoritmos a problemas en los que tendría una aplicación directa, como lo son los algoritmos de compresión relativa, tanto en el caso de secuencias biológicas como en el caso de trayectorias semánticas. También, aplicarlos en algoritmos como *Condensed Nearest Neighbour Data Reduction Algorithms* [12, 83], donde varias instancias son sustituidas por una, siendo la cadena media una candidata para la sustitución. De la misma forma, aplicarlo en *Synthetic Minority Oversampling Technique Algorithms* [84], es una idea similar a la anterior, donde también, a partir de instancias del conjunto, se construye una nueva instancia, salvo que en esta ocasión conservan en el conjunto tanto las instancias seleccionadas para generarla como la nueva instancia.

Bibliografía

[1] R. Mantegna, S. Buldyrev, A. Goldberger, S. Havlin, C.-K. Peng, M. Simons, and H. Stanley, "Systematic analysis of coding and noncoding dna sequences using methods of statistical linguistics," *Physical Review E*, vol. 52, no. 3, p. 2939, 1995.

[2] J. S. Richardson, B. Schneider, L. W. Murray, G. J. Kapral, R. M. Immormino, J. J. Headd, D. C. Richardson, D. Ham, E. Hershkovits, L. D. Williams, *et al.*, "Rna backbone: consensus all-angle conformers and modular string nomenclature (an rna ontology consortium contribution)," *Rna*, vol. 14, no. 3, pp. 465–481, 2008.

[3] J. K. Eng, A. L. McCormack, and J. R. Yates, "An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database," *Journal of the American Society for Mass Spectrometry*, vol. 5, no. 11, pp. 976–989, 1994.

[4] J. Onishi and T. Ono, "Contour pattern recognition through auditory labels of freeman chain codes for people with visual impairments," in *IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC)*, pp. 1088 – 1093, 2011.

[5] T. Kohonen, "Median strings," *Pattern Recognition Letters*, vol. 3, pp. 309–313, 1985.

[6] H. Bunke, X. Jiang, K. Abegglen, and A. Kandel, "On the weighted mean of a pair of strings," *Pattern Analysis & Applications*, vol. 5, pp. 23–30, 2002. 10.1007/s100440200003.

[7] X. Jiang, L. Schiffmann, and H. Bunke, "Computation of median shapes.," in *4th Asian Conf. on Computer Vision. Taipei, Taiwan*, pp. 300–305, 2000.

[8] N. Duta, A. K. Jain, and M.-P. Dubuisson-Jolly, "Automatic construction of 2d shape models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 5, pp. 433–446, 2001.

[9] A. Lourenço and A. Fred, "Ensemble methods in the clustering of string patterns," in *7th IEEE Workshops on Application of Computer Vision (WACV/MOTIONS)*, vol. 1, pp. 143 – 148, 2005.

[10] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, pp. 281–297, Oakland, CA, USA, 1967.

[11] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Syntethic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[12] P. Hart, "The condensed nearest neighbor rule (corresp.)," *IEEE transactions on information theory*, vol. 14, no. 3, pp. 515–516, 1968.

[13] T. Kohonen, "Self-organizing maps of symbols strings.," *Neurocomputing*, vol. 21, pp. 19–30, 1998.

[14] I. Fischer and A. Zell, "String averages and self-organizing map for strings," in *Proceedings of the Neural Computation 2000, Canada / Switzerland, ICSC*, pp. 208–215, Academic Press, 2000.

[15] R. Popovici and R. Andonie, "Sensor signal clustering with self-organizing maps," in *2015 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, July 2015.

[16] P. Somervuo, "Self-organizing map of symbol strings with smooth symbol averaging," in *Workshop on Self-Organizing Maps (WSOM'03). Hibikino, Kitakyushu, Japan (September 2003)*, 2003.

[17] J. Kruskal, "An overview of sequence comparison. time warps, string edits and macromolecules.," *SIAM Reviews*, vol. 2, pp. 201–2037, 1983.

[18] R. M. Ward, R. Schmieder, G. Highnam, and D. Mittelman, "Big data challenges and opportunities in high-throughput sequencing," *Systems Biomedicine*, vol. 1, no. 1, pp. 29–34, 2013.

[19] S. C. Ahalt, C. Bizon, and J. Evans, "Data to discovery: Genomes to health. a white paper from the national consortium for data science," tech. rep., University of North Carolina at Chapel Hill, 2014.

[20] B. Chern, I. Ochoa, A. Manolakos, A. No, K. Venkat, and T. Weissman, "Reference based genome compression," in *Information Theory Workshop (ITW), 2012 IEEE*, pp. 427–431, IEEE, 2012.

[21] P. Bille, P. H. Cording, I. L. Gørtz, F. R. Skjoldjensen, H. W. Vildhøj, and S. Vind, "Dynamic relative compression," *CoRR*, vol. abs/1504.07851, 2015.

[22] S. Kuruppu, S. J. Puglisi, and J. Zobel, *String Processing and Information Retrieval: 18th International Symposium, SPIRE 2011, Pisa, Italy, October 17-21, 2011. Proceedings*, ch. Reference Sequence Construction for Relative Compression of Genomes, pp. 420–425. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[23] S. Wandelt, M. Bux, and U. Leser, "Trends in genome compression," *Current Bioinformatics*, vol. 9, no. 3, pp. 315–326, 2014.

[24] R. Giancarlo, S. E. Rombo, and F. Utro, "Compressive biological sequence analysis and archival in the era of high-throughput sequencing technologies," *Briefings in Bioinformatics*, vol. 15, no. 3, pp. 390–406, 2014.

[25] R. W. Hamming, "Error detecting and error correcting codes," *The Bell system technical journal*, vol. 29, no. 2, pp. 147–160, 1950.

[26] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, pp. 707–710, 1966.

[27] F. J. Damerau, "A technique for computer detection and correction of spelling errors," *Commun. ACM*, vol. 7, pp. 171–176, Mar. 1964.

[28] D. Sankoff and J. B. Kruskal, eds., *Time warps, string edits and macromolecules: The theory and the practice of string comparison.* Addison-Wesley, 1983.

[29] D. Gusfield, *Algorithms on strings, trees, and sequences: computer science and computational biology.* New York, NY, USA: Cambridge University Press, 1997.

[30] G. Navarro, "A guided tour to approximate string matching," *ACM Comput. Surv.*, vol. 33, pp. 31–88, Mar. 2001.

[31] F. Casacuberta and M. Antonio, "A greedy algorithm for computing approximate median strings.," *VII Simposium Nacional de Reconocimiento de Formas y Análisis de Imágenes*, pp. 193–198, 1997.

[32] J. S. Sim and K. Park, "The consensus string problem for a metric is np-complete," *Journal of Discrete Algorithms*, vol. 1, pp. 111–117, 2003.

[33] F. Nicolas and E. Rivals, "Hardness results for the center and median string problems under the weighted and unweighted edit distances," *Journal of Discrete Algorithms*, vol. 3, no. 2-4, pp. 390–415, 2005. Combinatorial Pattern Matching (CPM) Special Issue. The 14th annual Symposium on combinatorial Pattern Matching.

[34] C. de la Higuera and F. Casacuberta, "Topology of strings: Median string is np-complete," *Theoretical Computer Science*, vol. 230, no. 1, pp. 39 – 48, 2000.

[35] D. Maier, "The complexity of some problems on subsequences and supersequences," *Journal of the ACM (JACM)*, vol. 25, no. 2, pp. 322–336, 1978.

[36] G. Sánchez, J. Lladós, and K. Tombre, "A mean string algorithm to compute the average among a set of 2d shapes.," *Pattern Recognition Letters*, vol. 23, pp. 203–213, 2002.

[37] X. Jiang, J. Wentker, and M. Ferrer, "Generalized median string computation by means of string embedding in vector spaces," *Pattern Recognition Letters*, vol. 33(7), no. 0, pp. 842–852, 2012.

[38] X. Jiang and H. Bunke, "Optimal lower bound for generalized median problems in metric spaces.," in *Structural, Syntactic, and Statistical Pattern Recognition*, vol. 2396, pp. 143–151, 2002.

[39] M. Hayashida and H. Koyano, "Finding median and center strings for a probability distribution on a set of strings under levenshtein distance based on integer linear programming," in *BIOSTEC*, pp. 108–121, 2016.

[40] M. Hayashida and H. Koyano, "Integer linear programming approach to median and center strings for a probability distribution on a set of strings," in *BIOSTEC*, pp. 35–41, 2016.

[41] X. Jiang, H. Bunke, and J. Csirik, "Median strings: A review," in *Data Mining in Time Series Databases* (M. Last, A. Kandel, and H. Bunke, eds.), pp. 173–192, World Scientific, 2004.

[42] C. Martínez-Hinarejos, A. Juan, and F. Casacuberta, "Median strings for k-nearest neighbour classification," *Pattern Recognition Letters*, vol. 24, no. 1-3, pp. 173–181, 2003.

[43] R. A. M. Cárdenas, "A learning model for multiple-prototype classification of strings.," in *17th Int. Conf. on Pattern Recognition (ICPR)*, vol. 4, pp. 420–423, 2004.

[44] J. Abreu and J. Rico-Juan, "A new iterative algorithm for computing a quality approximate median of strings based on edit operations," *Pattern Recognition Letters*, vol. 36, pp. 74 – 80, 2014.

[45] P. Mirabal, J. Abreu, and D. Seco, "Assessing the best edit in perturbation-based iterative refinement algorithms to compute the median string.," *Pattern Recognition Letters*, vol. 120, pp. 104–111, 2019.

[46] E. Chavez, S. Dobrev, E. Kranakis, J. Opatrny, L. Stacho, H. Tejeda, and J. Urrutia, "Half-space proximal: A new local test for extracting a bounded dilation spanner of a unit disk graph," in *Int. Conf. On Principles Of Distributed Systems*, vol. 1, pp. 235–245, 2005.

[47] F. Kruzslicz, "Improved greedy algorithm for computing approximate median strings.," *Acta Cybernetica*, vol. 14, pp. 331–339, 1999.

[48] C. D. Martínez-Hinarejos, A. Juan, F. Casacuberta, and R. Mollineda, *Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshops SSPR 2002 and SPR 2002 Windsor, Ontario, Canada, August 6–9, 2002 Proceedings*, ch. Reducing the Computational Cost of Computing Approximated Median Strings, pp. 47–55. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002.

[49] C. O. Rodríguez and J. O. Carratalá, "A stochastic approach to median string computation," *LNCS, SSPR&SPR*, vol. 5342, pp. 431–440, 2008.

[50] E. Ristad and P. Yianilos, "Learning string-edit distance," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 20, pp. 522 – 532, 1998.

[51] L. Franek and X. Jiang, "Evolutionary weighted mean based framework for generalized median computation with application to strings," in *Structural, Syntactic, and Statistical Pattern Recognition*, pp. 70–78, 2012.

[52] R. Wagner and M. Fischer, "The string-to-string correction problem.," *Journal of the ACM*, vol. 21, pp. 168–173, 1974.

[53] H. Freeman, "Computer processing of line-drawing data.," *Computer Surveys*, vol. 6, pp. 57–96, 1974.

[54] A. Jain and D. Zongker, "Representation and recognition of handwritten digits using deformable templates," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 19, pp. 1386 –1390, 1997.

[55] S. García-Díez, F. Fouss, M. Shimbo, and M. Saerens, "A sum-over-paths extension of edit distances accounting for all sequence alignments," *Pattern Recognition*, vol. 44, no. 6, pp. 1172 – 1182, 2011.

[56] J. R. Rico-Juan and J. M. I. nesta, "New rank methods for reducing the size of the training set using the nearest neighbor rule," *Pattern Recognition Letters*, vol. 33(5), no. 5, pp. 654 – 660, 2012.

[57] J. Rico-Juan and L. Micó, "Comparison of aesa and laesa search algorithms using string and tree-edit-distances," *Pattern Recognition Letters*, vol. 24, no. 9-10, pp. 1417–1426, 2003.

[58] J. G. Henikoff and S. Henikoff, "Blocks database and its applications," *Methods in Enzymology*, vol. 266, pp. 88 – 105, 1996.

[59] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín, "Searching in metric spaces," *ACM computing surveys (CSUR)*, vol. 33, no. 3, pp. 273–321, 2001.

[60] P. Ciaccia, M. Patella, F. Rabitti, and P. Zezula, "Indexing metric spaces with m-tree.," in *SEBD*, vol. 97, pp. 67–86, 1997.

[61] O. Pedreira and N. R. Brisaboa, "Spatial selection of sparse pivots for similarity search in metric spaces," in *International Conference on Current Trends in Theory and Practice of Computer Science*, pp. 434–445, Springer, 2007.

[62] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu, "Proximity matching using fixed-queries trees," in *Annual Symposium on Combinatorial Pattern Matching*, pp. 198–212, Springer, 1994.

[63] R. Baeza-Yates, "Searching: an algorithmic tour," *Encyclopedia of Computer Science and Technology*, vol. 37, pp. 331–359, 1997.

[64] M. L. Micó, J. Oncina, and E. Vidal, "A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements," *Pattern Recognition Letters*, vol. 15, no. 1, pp. 9–17, 1994.

[65] W. A. Burkhard and R. M. Keller, "Some approaches to best-match file searching," *Communications of the ACM*, vol. 16, no. 4, pp. 230–236, 1973.

[66] B. Bustos, O. Pedreira, and N. Brisaboa, "A dynamic pivot selection technique for similarity search," in *Data Engineering Workshop, 2008. ICDEW 2008. IEEE 24th International Conference on*, pp. 394–401, IEEE, 2008.

[67] S. Budalakoti, A. N. Srivastava, and M. E. Otey, "Anomaly detection and diagnosis algorithms for discrete symbol sequences with applications to airline safety," *IEEE Trans. on Systems, Man, and Cybernetics, Part C*, vol. 39, no. 1, pp. 101–113, 2009.

[68] R. Corral-Corral, E. Chavez, and G. Del Rio, "Machine learnable fold space representation based on residue cluster classes," *Computational biology and chemistry*, vol. 59, pp. 1–7, 2015.

[69] S. A. E. H. Mohamed, M. Elloumi, and J. D. Thompson, "Motif discovery in protein sequences," in *Pattern Recognition* (S. Ramakrishnan, ed.), ch. 1, pp. 5–18, Rijeka: IntechOpen, 2016.

[70] S. Kuruppu, S. J. Puglisi, and J. Zobel, "Relative lempel-ziv compression of genomes for large-scale storage and retrieval," in *String Processing and Information Retrieval - 17th International Symposium, SPIRE 2010, Los Cabos, Mexico, October 11-13, 2010. Proceedings*, pp. 201–206, 2010.

[71] C. Hoobin, S. J. Puglisi, and J. Zobel, "Relative lempel-ziv factorization for efficient storage and retrieval of web collections," *PVLDB*, vol. 5, no. 3, pp. 265–273, 2011.

[72] K. Liao, M. Petri, A. Moffat, and A. Wirth, "Effective construction of relative lempel-ziv dictionaries," in *Proceedings of the 25th International Conference on World Wide Web, WWW*

2016, Montreal, Canada, April 11 - 15, 2016, pp. 807–816, 2016.

[73] N. R. Brisaboa, T. Gagie, A. Gómez-Brandón, G. Navarro, and J. R. Paramá, "Relative compression of trajectories," *CoRR*, vol. abs/1810.05753, 2018.

[74] C. Parent, S. Spaccapietra, C. Renso, G. Andrienko, N. Andrienko, V. Bogorny, M. L. Damiani, A. Gkoulalas-Divanis, J. Macedo, N. Pelekis, Y. Theodoridis, and Z. Yan, "Semantic trajectories modeling and analysis," *ACM Comput. Surv.*, vol. 45, pp. 42:1–42:32, Aug. 2013.

[75] N. R. Brisaboa, G. de Bernardo, G. Navarro, T. V. Rodeiro, and D. Seco, "Compact representations of event sequences," in *2018 Data Compression Conference, DCC 2018, Snowbird, UT, USA, March 27-30, 2018*, pp. 237–246, 2018.

[76] M. Gagolewski, *Data fusion. Theory, Methods, and Applications*. Institute of Computer Science. Polish Academy of Sciences, 2015.

[77] X. Jiang, K. Abegglen, and H. Bunke, "Genetic computation of generalized median strings.," *Submitted for publication*, 2001.

[78] X. Jiang, K. Abegglen, H. Bunke, and J. Csirik, "Dynamic computation of generalised median strings," *Pattern Analysis & Applications*, vol. 6, pp. 185–193, 2003. 10.1007/s10044-002-0184-4.

[79] C. Martinez-Hinarejos, A. Juan, and F. Casacuberta, "Use of median string for classification," in *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, vol. 2, pp. 903 –906, 2000.

[80] J. K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang, "Distinguishing string selection problems," *Information and Computation*, vol. 185, no. 1, pp. 41–55, 2003.

[81] P. J. Van Laarhoven and E. H. Aarts, "Simulated annealing," in *Simulated annealing: Theory and applications*, pp. 7–15, Springer, 1987.

[82] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine learning*, vol. 3, no. 2, pp. 95–99, 1988.

[83] C.-H. Chou, B.-H. Kuo, and F. Chang, "The generalized condensed nearest neighbor rule as a data reduction method," in *18th International Conference on Pattern Recognition (ICPR'06)*, vol. 2, pp. 556–559, IEEE, 2006.

[84] J. Wang, M. Xu, H. Wang, and J. Zhang, "Classification of imbalanced data by using the smote algorithm and locally linear embedding," in *2006 8th international Conference on Signal Processing*, vol. 3, IEEE, 2006.