PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

Framework anytime para desarrollo de experimentos de algoritmos de optimización

ALEXIS SIMON ESPINOZA MENDOZA

ARIEL MARCOS MARCHANT GALLARDO

Profesor Guía: **Ignacio Araya Zamorano**

Profesor Co-referente: **Ismael Figueroa Palet**

Carrera**: Ingeniería Civil Informática**

Diciembre 2018

Índice

Resumen ..... **ii**
Abstract ..... **ii**
Lista de Figuras ..... **iv**
1. Introducción **1**
2.Objetivos proyecto **2**
2.1 Objetivo General **2**
2.2 Objetivos Específicos **2**
3. Estado del Arte **3**
3.1 Ingeniería de Algoritmos **3**
3.2 Comparación de Algoritmos **4**
3.3 Algoritmos Anytime **6**
3.4 Metodo Monte Carlo **6**
3.4.1 Sampleo Montecarlo……..……..6
4. Framework anytime para diseño de experimentos **7**
4.1 Semántica **7**
4.2 Estructura **8**
4.3 Algoritmo de gestión de procesos especulativo **9**
4.3.1 Función Select……...11
4.3.2 Función Retrive Node……12
4.3.3 Función Run……..13
4.3.4 Función Update……13
4.3.5 Función Best Strategy……14
4.4 Entrega de resultado anytime **15**
4.5 Paralelización automática **15**
5. Implementación **16**
6. Resultados Obtenidos **17**
7. Conclusión **19**
8. Referencias **20**

Resumen

Presentamos la propuesta de nuestra investigación, un *Framework anytime para el diseño de experimentos de comparación de algoritmos de optimización.* Este ámbito es de gran importancia en el área de investigación de operaciones, debido a la creciente aparición de nuevos algoritmos, se hace necesario la validación de estos mediante la comparación con otros algoritmos ya existentes, así como el análisis del efecto de los parámetros de los algoritmos en el comportamiento de los mismos. El *Framework* permite diseñar programas cuyo flujo está sujeto al resultado de la comparación de algoritmos, esto se logra mediante el algoritmo de ejecución especulativo que permite la ejecución del diseño experimental basado en los resultados parciales obtenidos por los algoritmos en un conjunto de instancias seleccionadas por el investigador. La calidad de los resultados obtenidos mejora respecto al tiempo, y se encuentra basado en el cálculo de probabilidades mediante el método Monte Carlo.

Palabras claves: Framework, diseño de experimentos, comparación algoritmos, optimización, anytime, especulativo, Método Monte Carlo

Abstract

We present the proposal of our research, an *anytime Framework for the design of experiments of comparison of algorithms of optimization*. This area is of great importance in the area of operations research, due to the growing appearance of new algorithms, it is necessary to validate these by comparing them with other existing algorithms, as well as the analysis of the effect of the parameters of the algorithms in the behavior of them. The Framework allows to design programs whose flow is subject to the result of the comparison of algorithms, this is achieved through algorithm of speculative execution, that allows the execution of the design of experiments based on the partial results obtained by the algorithms in the instances selected by the researcher. The quality of the results obtained improves with respect to time, and its based on the calculation of probabilities by Monte Carlo simulation.

Keywords: Framework, design of experiments, comparison algorithms, optimization, anytime, speculative, Monte Carlo Method.

Lista de Figuras

Figura 3.1 Ciclo ingeniería algoritmo 4
Figura 4.1 Best strategy 7
Figura 4.2 Árbol binario de probabilidad 9
Figura 4.3 Pseudocódigo gestión de procesos especulativo. 10
Figura 4.4 Diagrama flujo gestión de procesos especulativo. 11
Figura 4.5 Pseudocódigo función Select. 11
Figura 4.6 Pseudocódigo función Retreve Node 12
Figura 4.7 Pseudocódigo función Run 13
Figura 4.8 Pseudocódigo función Update 14
Figura 6.1 Script diseño experimental 17
Figura 6.2 Grafico calidad solución v/s n° instancias primera prueba 18
Figura 6.3 Grafico calidad solución v/s n° instancias segunda prueba 18

1. Introducción

En el contexto de la investigación de las ciencias de computación, los investigadores se ven enfrentados a circunstancias de crear y desarrollar un nuevo algoritmo para resolver una necesidad o problemática; es aquí donde es necesaria la ingeniería de algoritmos como una guía para la creación y modificación de algoritmos. En esta disciplina se distinguen distintas fases del proceso que intervienen en el desarrollo de algoritmos como lo son el diseño, análisis, implementación y experimentación. Nuestro objetivo es aliviar las actividades comunes desarrolladas por los investigadores mediante la creación de un *Framework anytime, para desarrollo de experimentos,* entregando una asistencia para el investigador durante la fase de experimentación.

En el diseño de experimentos se desea proponer un flujo de control, sujeto a la comparación tanto de dos algoritmos como un grupo de estos entre sí. Entregando una herramienta para el investigador que permita expresar de manera sencilla las ejecuciones de los algoritmos que están involucrados en la fase de experimentación.

Además, el *Framework anytime para diseño de experimentos* podrá entregar resultados preliminares durante la ejecución de los experimentos. Las aplicaciones anytime pueden entregar un resultado válido antes de su finalización. Esta característica permitirá acelerar la toma de decisiones en la comparación de algoritmos, ya que no es necesario esperar hasta finalización del experimento para obtener información relevante de las comparaciones.

Para la experimentación se busca la agilización en la entrega de resultados en la comparación de algoritmos, mediante la creación de una política de procesamiento, que otorgue resultados de comparación cuya calidad se maximiza con el paso del tiempo. Esto se logra con el cálculo estadístico de probabilidades, basado en el método de Monte Carlo.

En el presente informe se expone la entrega final a la propuesta del *Framework anytime para desarrollo de experimentos,* donde se muestra los módulos que lo componen. Como los son la *semántica, gestión de proceso especulativo, resultados anytime y paralelización automática*. Además de describir los resultados obtenidos con las pruebas finales realizadas y actualización del estado del arte con nueva literatura involucrada y sus referencias respectivas.

2.Objetivos proyecto

2.1 Objetivo General

Implementar *Framework anytime para diseño de experimentos* que ayude a diseñar y agilizar la etapa de experimentación para poder hacer comparación de algoritmos de optimización, permitiendo que la toma de decisiones sea en menor tiempo posible

2.2 Objetivos Específicos

Dentro de los objetivos específicos del proyecto se encuentran los siguientes:

a. Diseñar el *Framework.*

b. Investigar e Implementar Sampleo de Parámetro de Monte Carlo

c. Investigar e Implementar paralelización automática

d. Testear *Framework* con casos reales.

e. Analizar resultados obtenidos sobre las comparaciones algoritmos.

3. Estado del Arte

En la literatura existen distintos trabajos que muestran y describen los herramientas y técnicas que se relacionan al proyecto. En esta sección describiremos el estado del arte relacionado:

- Ingeniería de Algoritmos
- Comparación Algoritmos
- Algoritmos Anytime
- Método Monte Carlo

3.1 Ingeniería de Algoritmos

Ingeniería de algoritmos es el estudio asociado para el desarrollo de algoritmos, que tiene como objetivo principal el poder acelerar la transferencia de conocimientos algorítmico, mediante la estandarización de las distintas fases de desarrollo de algoritmos, las que se definen como ciclo Ingeniería Algoritmo. Obteniendo como beneficio la robustez y el buen rendimiento de los algoritmos sumando a esto la facilidad en la evaluación experimental para satisfacer las demandas prácticas. Entregando una gran ventaja para la comparación entre pares y futuras mejoras, ya que se vuelven más comprensible para los desarrolladores de algoritmos.

El ciclo Ingeniería Algoritmo posee un total 9 fases, mostrado en el modelo (*Figura 3.1 Ciclo Ingeniería Algoritmo*) propuesto por Sanders[1]. Este ciclo es descrito inicialmente con la **aplicación** específica que se quiere desarrollar (1), donde se crea un **modelo realista** (2) que se va enfrentar el algoritmo a desarrollar, permitiendo la comparación de los resultados obtenidos con los del modelo realista y así visualizar coincidencia entre estos. Posteriormente comienza el ciclo de la ingeniería de algoritmos por su fase inicial **diseño algoritmo** (3) para plantear el modelo solución que va cumplir el algoritmo. Luego sigue a la fase de **análisis** (4) la que, desde un punto de vista teórico en relación al diseño planteado, permite obtener las **garantías de rendimiento** (5), como tiempo de ejecución asintótico, aproximación, etc.

Luego de las fases anteriores se procede a la **implementación** (6) del algoritmo propuesto, etapa que no se debe subestimar, ya que, en ella se construye el algoritmo como tal para que cumpla las necesidades requeridas por la aplicación. Terminando la implementación prosigue la fase de **experimentación** (7), donde se realiza todas las observaciones correspondientes, haciendo que sea una pieza fundamental y dar término del ciclo. Esta necesita de datos de **entrada reales** (8), que deben ser análogos al modelo realista planteado anteriormente. Los beneficios que genera la experimentación son numerosos, como lo son la estimación de constantes que están involucradas y no fueron encontradas en el análisis, asimismo es posible encontrar el comportamiento del algoritmo de forma intuitiva y desde ese punto realizar distintas comparaciones con algoritmos similares.

Terminando el ciclo de la ingeniería del algoritmo, con los conocimientos obtenidos en las fases de diseño análisis, implementación y experimentación, es necesario encontrar nuevos diseños de algoritmos mejorados, es por eso que se menciona que este ciclo de ingeniería de algoritmo es iterativo.

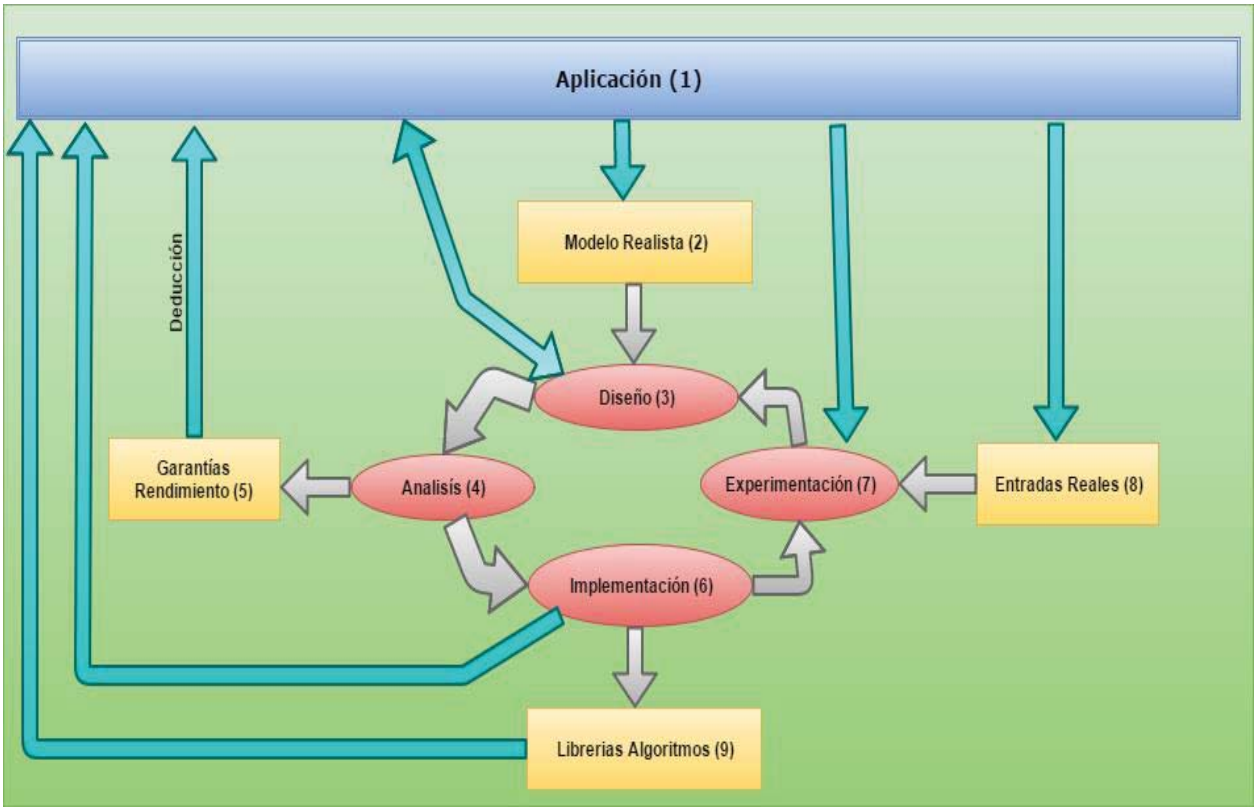

Figura 3.1 Ciclo ingeniería algoritmo

Otro subproducto del ciclo de la ingeniería de algoritmo es **biblioteca de algoritmo** (9), es decir, el código en cual se implementó debe escribirse de forma que este pueda ser reutilizable, de modo que puedan usarse posteriormente para un desarrollo futuro. Esto ocurre en las comunidades de programación, donde cada vez se encuentran nuevas librerías y plugins que utilizan algoritmos facilitando las tareas para los programadores o investigadores en sus nuevas aplicaciones.

3.2 Comparación de Algoritmos

En [2] se explican lineamientos generales en la comparación de metaheurísticas, sin embargo, estos lineamientos pueden ser considerados para la comparación de algoritmos de optimización en general.

El primer aspecto a ser tomado en cuenta es lo que se conoce como **testbed**, es decir bajo qué circunstancias se comparan los algoritmos. El caso ideal es el uso de instancias de problemas ya utilizados en publicaciones previas. Esto permite obtener resultados comparables entre dos algoritmos. En el caso de no existir instancias para el problema que se resuelve, es necesaria la creación de nuevas instancias, que cumplan con ser similares a situaciones de la vida real, que tengan varios niveles de dificultad. Esto concede la posibilidad de analizar el comportamiento del algoritmo en aspectos de calidad de solución y tiempo de ejecución, en distintas situaciones y permite discriminar qué método funciona mejor para cada situación. Se debe tener en cuenta también la accesibilidad de las instancias creadas para futuros investigadores, y esta

accesibilidad debe ser en forma de la publicación de las instancias mismas o una función generadora de instancias.

Comúnmente un algoritmo posee lo que se conoce como parámetros o hiperparametros, esto son componentes configurables que cambian la performance de un algoritmo, son establecidos antes de la ejecución. Estos parámetros pueden ser fijos o estar en función de la instancia utilizada. En ambos casos deben ser predeterminados por el diseñador. Ejemplos de parámetros son el número de individuos y la tasa de mutación en un algoritmo genético. El segundo aspecto a considerar es el número de parámetros, en general el número de parámetros de un algoritmo refleja la complejidad del mismo y siempre es más preferible la simpleza ya que un algoritmo simple es fácil de implementar explicar y analizar. La sintonización de parámetros o **parameter tuning** busca encontrar dentro del espacio de posibles combinaciones de parámetros, aquel conjunto que produzca la mejor performance para el algoritmo analizado[7].

Finalmente es crucial medir empíricamente la calidad de las soluciones de los algoritmos a comparar, para ello usualmente se heredan las métricas utilizadas por otros autores. Se necesita comprobar si las diferencias observadas durante la experimentación son producto del azar o están realmente presentes.

Si se desea comprobar estas diferencias se debe realizar un test estadístico[10]. Esto significa que una hipótesis nula, la cual declara que no existe diferencia entre los grupos estudiados, debe ser formulada, y además se debe identificar un test estadístico adecuado.

Luego el test estadístico calcula la probabilidad de obtener los datos observados, dado que la hipótesis nula es correcta. Un p-value pequeño significa que esta probabilidad es leve. La hipótesis nula es rechazada si el p-value es menor que un nivel de significancia definido anteriormente. Un estadístico es calculado a partir de los datos observados y forma la base de la prueba estadística. Si se hacen suposiciones acerca de la distribución de los datos, la distribución teórica del estadístico. El valor de la variable de test calculado mediante las observaciones es comparado con la distribución esperada, si la hipótesis nula es correcta, si este valor es mayor o menor que un límite especificado, entonces es poco probable que la hipótesis nula es correcta y por lo tanto es rechazada. Finalmente, el resultado tiene un nivel de significancia estadística alfa.

- **T-test:** También conocido como prueba t de Student, es un test para datos continuos. Se utiliza para averiguar si los valores esperados de dos grupos son iguales, asumiendo que los datos están distribuidos normalmente. Este test puede ser utilizado ya sea para grupos pareados o no pareados.

- **F-test:** Prueba de f de Fisher, que utiliza de la distribución de f de Fisher para comprobar si la hipótesis nula no puede ser rechazada mediante en que dos muestras presentan igualdad de varianza y así se contrasta la normalidad.

- **ANOVA:** Análisis de varianza, es una técnica estadística que señala si dos variables están relacionadas por sus medias, se utiliza para esto un factor de ANOVA para decir que tan diferentes son la media de una variable con las de la población o grupos de otra variable.

- **Wilcoxon-test:** Test para datos continuos u ordinales. En contraste a test t de Student, este no requiere que los datos están normalmente distribuidos. Este test también puede ser utilizado para datos pareados o no pareados.

3.3 Algoritmos Anytime

Los algoritmos *anytime* son algoritmos cuya calidad de resultados mejora gradualmente en la medida que el tiempo de computación incrementa. El tiempo de computación necesario para calcular soluciones óptimas o precisas, típicamente disminuye la utilidad del sistema. Por ello es beneficioso construir sistemas que puedan canjear calidad de resultados por costo de computación. Esto permite crear procedimientos que pueden retornar varias aproximaciones posibles.

Una característica esencial de los algoritmos anytime es la definición de métricas de calidad, para monitorear el progreso de la solución del problema, ya sea midiendo el grado de certeza de que el resultado es correcto, o que tan cerca esté del resultado real. Además, un algoritmo anytime idealmente debería contar con varias de las siguientes propiedades:

- La calidad de un resultado aproximado puede ser determinado de manera precisa.
- La calidad de la solución aumenta siempre en el tiempo.
- El algoritmo puede ser detenido en cualquier momento de la ejecución y proveer alguna solución; el algoritmo puede pausarse y reanudarse[3].

3.4 Método Monte Carlo

El método *Monte Carlo* es un método estadístico que permite resolver problemas matemáticos mediante la simulación de variables aleatorias[4]. Este método hace uso del muestreo repetido de variables aleatorias, para obtener resultados acerca de la distribución de estas variables, para luego poder hacer cálculos de probabilidad a partir de la misma. Los pasos para aplicar el método *Monte Carlo* son generar una serie de números aleatorios, utilizar esta secuencia para generar otra secuencia, según sea el caso de uso de la problemática a resolver, finalmente la secuencia obtenida es utilizada para estimar una propiedad del sistema simulado, específicamente el cálculo de probabilidades a partir del radio entre las ocurrencias en una región del sistema y las ocurrencias totales.

3.4.1 Sampleo Montecarlo

En estadística bayesiana se puede obtener la distribución del parámetro(s) de una variable aleatoria dado una muestra y la distribución que se cree que llevan. Están distribuciones son conocidas como posterior distribution y prior distribution específicamente. Estos modelos permiten obtener tanto probabilidad de que un parámetro tenga cierto valor, como hacer un sampleo de los parámetros de la variable estudiada. para encontrar la distribución posterior y poder hacer una sampleo de los parámetros, existen técnicas conocidas como Markov Chain Monte Carlo o MCMC[12].

4. Framework anytime para diseño de experimentos

El propósito de la etapa de experimentación es obtener resultados que nos indiquen el desempeño de un algoritmo propuesto en comparación a otro, en términos de calidad de solución o tiempo, con un conjunto dado de instancias. En la medida en que se ponen a prueba los algoritmos en cada instancia los resultados de estos son utilizados para entrar en la simulación de *Monte Carlo* que nos indican si un algoritmo será o no mejor que el otro con el conjunto de instancias dado.

En esta sección se describe el Framework para diseño de experimentos propuesto y sus principales componentes, los cuales son:

- Semántica
- Estructura
- Gestión de procesos especulativo
- Entrega de resultados anytime
- Paralelización automática

4.1 Semántica

El *Framework anytime para diseño de experimentos* propuesto proporciona una semántica, que contiene distintos componentes y funciones que la conforman. Cuando deseamos comparar dos algoritmos podemos hablar de que estamos poniendo a prueba dos estrategias distintas para resolver un problema, por ello como primer componente de la semántica encontramos el objeto **Strategy** (S), este objeto consta de la tupla algoritmo-configuración, donde **Algoritmo** es el conjunto de datos que lo definen como el nombre y la ruta hacia el programa ejecutable, en cuanto a la configuración, está es un conjunto de parámetros y pueden contar con un valor por defecto.

En segundo lugar, se proporciona la definición del **conjunto de instancias** (π) que es la ruta donde encontramos el archivo que define el conjunto de instancias con las que se ejecutarán posteriormente las distintas estrategias puestas a prueba en el experimento.

La semántica permite el uso de sentencias y control de flujo de un lenguaje estructurado común (if, else, while, for, print, declaración de variables, operadores matemáticos y lógicos), pero su núcleo está basado en la función bestStrategy. Que dadas dos estrategias retorna aquella que supera a la otra, en el conjunto de instancias (π) o un **rango** (R) de este**.** Lo que puede ser expresada de la siguiente forma. (*Figura 4.1 Best strategy).*

```
Strategy <-- best_strategy(S, S', π, R, delta)
```

Figura 4.1 Best strategy

La cual retorna una **Strategy** (S o S') que supere significativamente a la otra estrategia dentro del rango especificado del conjunto π o total de este.

Sumado a esto se permiten las siguientes expresiones:

- ***Strategy best_strategy(S_list, pi, delta_sig)***, Similar a la anterior, pero recibe una lista de algoritmos, de este modo, se facilita el diseño y se automatiza el orden de las comparaciones. Retorna la mejor estrategia de la lista de estrategias dada.

- ***double best_parameter_value(S, param, values, pi, delta_sig),*** Dada una estrategia, y su conjunto de parámetros, retorna el mejor valor encontrado para el parámetro **param** de la estrategia **S** (siempre y cuando el nuevo valor mejore significativamente al valor por defecto). Donde **values** es una lista de valores que puede tomar el parámetro.

4.2 Estructura

El *Framework anytime para diseño de experimentos* posee 2 estructuras datos principales, por un lado, se encuentra un **árbol binario de probabilidad** que almacena los distintos nodos de comparación de algoritmos y por otro lado se encuentra **matriz de resultados**, donde se almacena los resultados obtenidos de los algoritmos y una serie de instancias asociada. Con estas estructuras es posible dar vida al *Framework anytime para diseño de experimentos*, permitiendo que pueda cumplir las necesidades por la que ha sido desarrollado.

El **árbol binario de probabilidad** es desarrollado a partir de la configuración descrita por el usuario por la semántica previamente descrita. Representa el conjunto completo de posibles comparaciones entre algoritmos de acuerdo a esa configuración. Los *nodos* representan las comparaciones entre dos algoritmos (A1 y A2), el subárbol izquierdo de un nodo corresponde al conjunto posible de comparación dado que A1 es mejor que A2; el subárbol derecho es análogo, pero asumiendo que A2 es mejor que A1. Está comparación va ser reflejada por las ramas, las cuales representan un porcentaje de probabilidad en que un algoritmo es mejor a otro. Terminando por hojas las que representan las comparaciones finales (Mostrando que algoritmo es mejor), los cuales podrían ser los obtenidos al realizar todas las instancias o por entrega de resultados anytime donde el resultado va terminar siendo un nodo hoja en el momento en que fue solicitado.

La siguiente figura (*Figura 4.2 Árbol binario de probabilidad)* representa una ejemplificación de lo descrito anteriormente con la comparación de 3 algoritmos. En la que aparecen 2 nodos de comparación de color celeste y 3 nodos hojas color verde, los cuales poseen un número en su interior de color azul, este número representa la probabilidad condicional. El nodo raíz siempre será 100 en probabilidad condicional, luego de aplicar las simulaciones de Monte Carlo se obtienen un valor de simulaciones representado en los porcentajes de las ramas, entonces para tener la probabilidad condicional es necesario realizar una división entre simulación divido con probabilidad condicional del padre. Entonces de esta forma se actualizando el árbol a medida que se van agregando instancias. Una vez que ha solicitado el usuario resultados, es posible que entregue el mejor nodo hoja hasta el momento, ósea el que tenga mayor probabilidad condicional, que en la figura (*Figura 4.2 Árbol binario de probabilidad)* es el algoritmo 2 por sobre los demás.

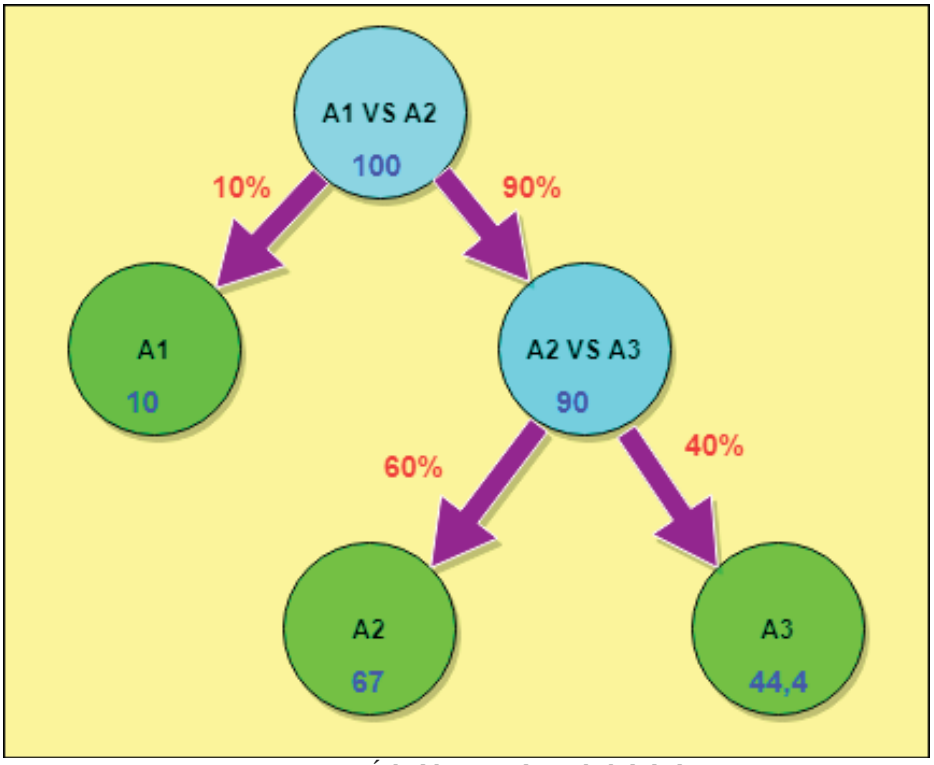

Figura 4.2 Árbol binario de probabilidad

Adicionalmente se utiliza una estructura de datos de una **matriz de resultados** MxN, donde M corresponde a la cantidad total de algoritmos y N es la cantidad total de instancias. Cada casilla (i,j) de la matriz almacena el resultado (evaluación) retornado por una ejecución del algoritmo i en la instancia j. Con esta matriz es posible mantener los resultados y realizar las distintas funciones para la actualización de comparación según lo requiera *Framework anytime para diseño de experimentos*.

4.3 Algoritmo de gestión de procesos especulativo

Este algoritmo es el encargado de realizar las experimentaciones de manera sistemática, de tal manera de que se pueda obtener información relevante en poco tiempo, permitiendo de esta manera que el investigador pueda tomar decisiones respecto a los resultados obtenidos. Las ejecuciones están organizadas en un **árbol binario de probabilidad** descrito anteriormente.

Inicialmente el **árbol**, se encuentra vacío y al comenzar la experimentación el **árbol** comienza agregar nodos no repetidos según diseño propuesto por el investigador, ya que el *Framework anytime para diseño de experimentos,* mediante a la función *Best_strategy* descrita anteriormente genera una nueva comparación, es decir la cantidad de funciones *Best_strategy* propuestas, será la cantidad mínima de nodos de comparación que tendrá el **árbol.**

En cada iteración, la función **Select** es la encargada de seleccionar el siguiente nodo a procesar. **Run** ejecuta los algoritmos dentro del nodo seleccionado en instancias escogidas aleatoriamente. Terminando con la función **Update** donde se calculan y actualizan las probabilidades de todos los nodos del **árbol binario de probabilidad,** para así con las nuevas probabilidades aplicar nuevamente la función **Select** para seleccionar el mismo nodo u otro para

seguir ejecutando los algoritmos con nuevas iteraciones. Esto se demuestra en la siguiente figura (*Figura 4.3 Pseudocódigo gestión de procesos especulativo*).

```
Function SpeculativeExecute():
      create node set S and include root node
      while S is not empty do
            n <~ select(S)
            if not n
                  break
            run(n)
            update(S)
```

Figura 4.3 Pseudocódigo gestión de procesos especulativo.

Como condición término para este monitor de proceso, es cuando la función **Select** no retorne el mejor nodo para ejecutar una iteración y esto puede ocurrir ya que se ejecutaron todas las instancias posibles para cada nodo. Cada iteración del algoritmo va de la mano con una ejecución del experimento diseñado, recorriéndolo según los resultados obtenidos hasta ese momento.

La siguiente figura (*Figura 4.4 Diagrama flujo gestión de procesos especulativo*) se muestra un diagrama de flujo del algoritmo de gestión de proceso especulativo, donde se destacan las funciones mencionadas en el pseudocódigo y como ocurren las transiciones en el ciclo a medida que se genera ejecuciones. Además, aparecen otras funciones como lo son **Retrieve Node** y **Best Strategy** que son relevantes para el funcionamiento de gestión de procesos especulativo. A continuación, se describe con mayor detalle todas estas funcionen que componen el algoritmo.

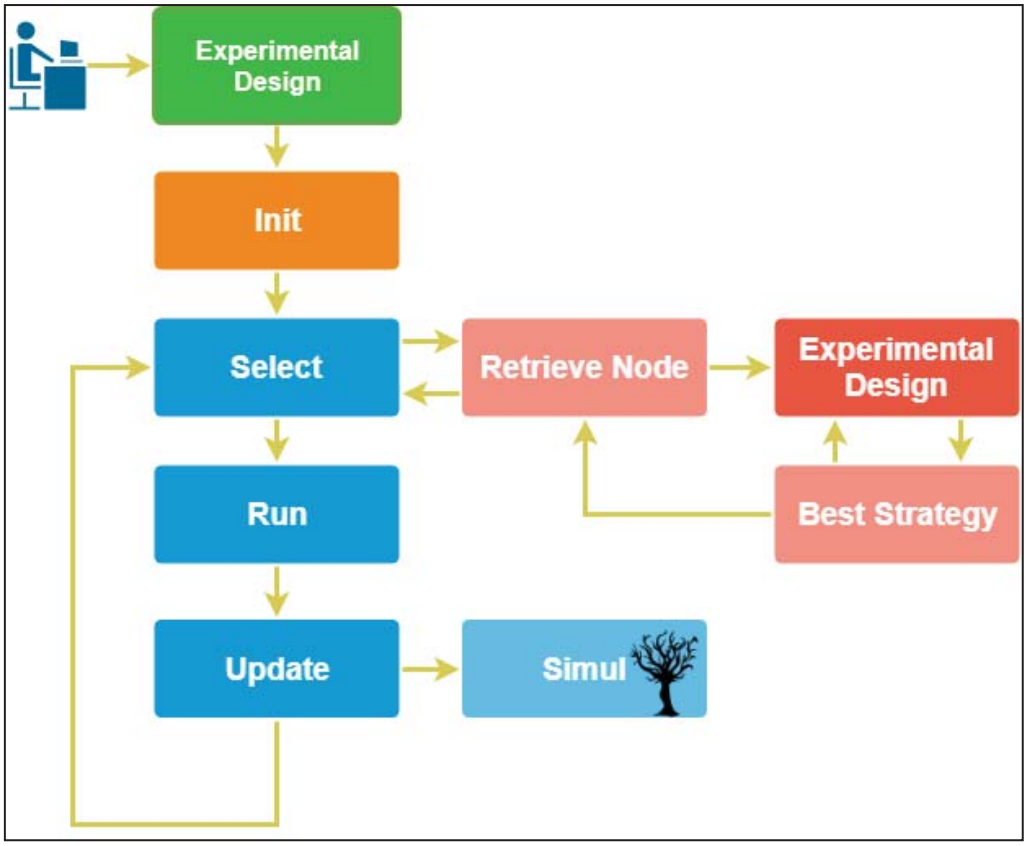

Figura 4.4 Diagrama flujo gestión de procesos especulativo

4.3.1 Función Select

La función **Select** es la encarga de seleccionar mejor nodo a ejecutar, es efectuada recorriendo árbol de la raíz hasta las hojas, a la vez encuentra el nodo que minimiza la probabilidad condicional en la mejor rama encontrada. La decisión para elegir el camino está basada en el número de simulación de los nodos hijos efectuada durante la iteración anterior, de manera que un nodo con mayor número de simulaciones es mejor respecto a su nodo hermano. El nodo que minimiza probabilidad condicional es el candidato a ejecutarse con una nueva instancia, sin embargo, si a partir del nodo hoja se puede agregar un nuevo nodo desde el diseño experimental, este nuevo nodo agregado será el próximo a ejecutarse. Puede retornar nulo en el caso el mejor nodo seleccionado ya hubiera ejecutado todas sus instancias y con esto terminaría la experimentación. La siguiente figura (*Figura 4.5 Pseudocódigo función Select*) representa el pseudocódigo que describe la función **Select.**

```
Function Select(S):
      selected <~ None
      aux <~ root
      while aux is not null:
            if aux is minimum
                  selected = aux
            if aux.left.simulations > aux.right.simulations
                  aux <~ aux.left
            else
                  aux <~ aux.right.simulations
      nod <~ retrieveNode(aux)
      if nod is not null:
            add(nod)
      if selected is not terminated:
            return selected
      else
            return None
```

Figura 4.5 Pseudocódigo función Select.

4.3.2 Función Retrieve Node

El propósito de esta función es agregar un nuevo nodo al árbol utilizando la información de los resultados preliminares, es llamada en la selección de mejor nodo si el nodo con mayor probabilidad conjunta puede agregar un hijo en el camino correcto. Las variables **mensaje** (msg), **contador**(count) y **speculativeNode**, son globales. Cada nodo puede generar un mensaje del tipo binario [0,1,1,0,...] que indica el camino desde el nodo raíz hasta sí mismo, se forma con el resultado actual de la comparación de los algoritmos del nodo donde 0 significa que el primer algoritmo le ganó al segundo, es decir que proviene por la rama izquierda de su nodo padre, y 1 lo contrario. Con la información del mensaje podemos recorrer especulativamente el experimento diseñado por el usuario. Se hace uso de la sentencia try debido a que en las primeras iteraciones el árbol aún no tiene su forma completa y por ende no se tienen resultados suficientes para recorrer el experimento por completo, siendo necesario interrumpir en cierta comparación para retornar el nuevo nodo correspondiente. La siguiente figura (*Figura 4.6 Pseudocódigo función Retreve Node*) representa el pseudocódigo que describe la función **Retrieve Node.**

```
Function retrieveNode(node):
      msg<~node.msg()
      try:
            count <~0
            speculativeNode <~ null
            experimentalDesign()
      return speculativeNode
```

Figura 4.6 Pseudocódigo función Retreve Node

4.3.3 Función Run

La función **Run** recibe un nodo como parámetro, permitiendo la ejecución de los algoritmos que se encuentran en el nodo con una instancia aleatoria no seleccionada antes por este nodo y se almacena el resultado obtenido (valor retorno del algoritmo) el cual es almacenado en la matriz de resultados globales. Si el nodo es primera vez que es seleccionado, pasa a un estado visitado y se ejecutan 3 instancias aleatorias, para así obtener datos iniciales y así poder calcular las probabilidades respectivas. Como condición de término para esta función, se tiene que, al seleccionar la última instancia existente se deja que la probabilidad quede como 1 para el algoritmo vencedor, de este modo el nodo no será seleccionado nuevamente.

También la función **Run** es relacionada al componente de paralelización automática del *Framework anytime para diseño de experimentos*, donde se ejecutan distintas funciones **Run** por cada core que posea el procesador del equipo, permitiendo la ejecución de más instancias en un menor tiempo. La siguiente figura (*Figura 4.7 Pseudocódigo función Run*) representa el pseudocódigo que describe la función **Run.**

```
Function Run(n):
      if n is visited
            i <~ selectInstance(n)
            execute algorithms in n with and instance i
      else
            execute algorithms in n with three instances
```

Figura 4.7 Pseudocódigo función Run

4.3.4 Función Update

La función **Update** tiene como objetivo actualizar las estadísticas del árbol, basándose en el muestreo de simulaciones Monte Carlo. Cada simulación es un recorrido desde la raíz del árbol, bajando por este según la comparación entre la suma del resultado de la ejecución de todas las instancias del primer y segundo algoritmo del nodo, hasta llegar a un nodo hoja.

Cada nodo visitado en el recorrido añade una visita, que sirve para calcular la probabilidad condicional utilizada en la selección de mejor nodo. Durante el recorrido deseamos comparar la suma total de los resultados de los algoritmos en cada nodo, más un umbral:

$$\sum_{i=1}^{N} X_i + \Delta \times N > \sum_{i=1}^{N} Y_i \quad (4.2.4.1)$$

El uso del umbral es debido a que solo desechamos el primer algoritmo en el caso de que el segundo algoritmo sea mejor que esté, más un valor arbitrario. Podemos escribir la suma total como:

$$\sum_{i=1}^{N} X_i = \sum_{i=1}^{n} X_i + \sum_{i=n}^{N} X_i \qquad (4.2.4.2)$$

Donde el primer sumando es la suma de los resultados obtenidos hasta el momento (conocido) y el segundo sumando la suma de los resultados faltantes.

Debido a que no se poseen los resultados faltantes, podemos hacer un muestreo de la misma mediante la generación de un numero aleatorio que siga una distribución normal. Se sabe que la suma ***sum*** de una variable aleatoria sigue una distribución normal:

$$\sum_{i=1}^{N} X_i \quad \text{sum} \sim N(\, n \times x \,,\, n^2 \times sd) \quad (4.2.4.3)$$

Los valores de la media **x** y desviación estándar **sd** son muestreados a partir de la distribución posterior de los parámetros de los datos, esta distribución se obtiene mediante la técnica Markov Chain Monte Carlo[12].

Una vez terminadas las simulaciones poseemos en cada nodo un número de simulaciones con el cual podemos calcular la probabilidad condicional del resultado de una comparación como la razón entre las simulaciones propias y las del nodo padre:

$$\text{Pcond (nod)} = \frac{nodo\ visitas}{nodo\ padre\ visitas} \quad (4.2.4.4)$$

La siguiente figura (*Figura 4.8 Pseudocódigo función Update*) representa el pseudocódigo describe la función **Update.**

```
Function Update(root):
      root.refreshSimulations()
      for x in 1:100:
            simulation(root)
```

Figura 4.8 Pseudocódigo función Update

4.3.5 Función Best Strategy

Las Funciones **experimentalDesign** y **Best Strategy** van de la mano, según el largo del mensaje, se avanza especulativamente por la función de diseño experimental. Pueden ocurrir dos cosas, si el largo del mensaje es menor al número de llamadas a **BestStrategy**, se interrumpirá el recorrido especulativo y se retornará un nuevo nodo, con las estrategias correspondientes para agregar al árbol. La siguiente figura (*Figura 4.9 Pseudocódigo función Best Strategy*) representa el pseudocódigo describe la función B**est Strategy.**

```
Function bestStrategy(S1,S2,m):
      if count<len(msg)
            if msg[count]==0
                  count ++
            return S1
            else
                  count ++
                  return S2
      speculativeNode<~new Node(S1,S2,m)
      raise error(“return node”)
```

Figura 4.9 Pseudocódigo función Best Strategy

4.4 Entrega de resultado anytime

El resultado entregado es la ejecución del programa escrito por el usuario, que está sujeto a las comparaciones escritas como llamadas a la función *Best strategy*. Esta ejecución es especulativa, debido a que el resultado de la comparación es preliminar y está basada en los resultados obtenidos hasta el momento actual y la simulación de los resultados faltantes.

El *Framework anytime para diseño de experimentos* entrega resultados cuya calidad mejora en el tiempo, por ello debemos definir la métrica utilizada para medir la calidad de la solución entregada. La métrica que me dirá la calidad es la solución entregada en un momento cualquiera es la máxima probabilidad condicional del conjunto de nodos hojas del **árbol binario de probabilidades**, representa la probabilidad de que el resultado obtenido sea el correcto. Esta métrica cumple con las propiedades necesarias del *anytime* mencionadas en puntos anteriores.

4.5 Paralelización automática

La paralelización automática permite convertir código secuencial en multihilo con el objetivo de utilizar la mayor cantidad de recursos posibles del equipo, es decir, la utilización de los múltiples procesadores, de forma simultánea en el dispositivo. Esto se encuentra reflejado en los core que tienen los procesadores que componen los notebook o computadores actualmente.

El *Framework anytime para diseño de experimentos* puede ejecutar de manera concurrente varias instancias en los algoritmos, durante la fase de ejecución. Esto funciona, ya que, es posible obtener la cantidad de procesadores disponibles que tenga el equipo, permitiendo de esta forma utilizar la totalidad de estos y repartir a cada procesador una ejecución de un algoritmo con una respectiva instancia, de manera equitativa para no genera una ventaja entre un algoritmo versus el otro. Entonces a medida que aumente la cantidad de procesadores, menor será el tiempo de ejecución de la totalidad de instancias. Entregado mejores resultados en un corto tiempo.

5. Implementación

El *Framework anytime para diseño de experimentos* es desarrollado en lenguaje de programación Python (versión 3.6) con un Pip (versión 18.1). Este lenguaje posee una serie de librerías necesarias para cumplir con los requerimientos que debe satisfacer esta aplicación. Sumando de tener ventajas de simplificación, flexibilidad y orden en la programación. Por lo tanto, si un investigador desea utilizar el *Framework* son requisitos tener instalado este lenguaje de programación en su versión mencionada anteriormente.

Actualmente el *Framework anytime para diseño de experimentos* solo ha sido testeado en sistema operativo **Linux Ubuntu 16.04 y Windows 10,** teniendo un buen rendimiento**.** También se busca hacer que pueda ser un *Framework* multiplataforma, es decir que pueda ser utilizado en cualquier sistema operativo.

Entre las librerías utilizadas de Python se encuentra es Scipy que es una librería del campo de la ciencia, que se caracteriza por el uso de las estadísticas. Entregando ventajas para poder calcular los distintos test estadísticos que involucran al *Framework anytime para diseño de experimentos* y así obtener valores de las probabilidades de que cual algoritmo es mejor que los demás

Otra librería utilizada es Matplotlib que es una librería relacionada a la visualización, que se caracteriza por la generación de gráficos. Esencialmente para el tema de resultados anytime, de esta forma el usuario pueda visualizar cómo van evolucionando las probabilidades de calidad de solución de los algoritmos respecto a las ejecuciones realizadas. Además, es posible con los gráficos encontrar ciertas tendencias. El *Framework anytime para diseño de experimentos* actualmente entrega gráficos ***% calidad solución v/s ejecución***, señalando que algoritmo tiene mayor porcentaje por cada ejecución.

El *Framework anytime para diseño de experimentos* posee un archivo README.txt, donde se muestran los requerimientos necesarios para utilizarlo, comando de instalación para los distintos sistemas operativos y el script de diseño de experimentos estándar, donde el investigador diseñe las distintas estrategias que va a probar.

6. Resultados obtenidos

Se elaboraron distintas pruebas para analizar el comportamiento del *Framework anytime para diseño de experimento* mediante un **Dataset** de 1600 instancias, de un problema de optimización. Con estos datos se simula la ejecución del experimento para la comparación de un algoritmo y sus distintas versiones, cada una con distintos parámetros. El script de diseño experimental, descrito en la semántica se muestra en la siguiente figura (*Figura 6.1 Script diseño experimental*).

```
def experimentalDesign():

  params = {"__a" : 0.0, "__b" : 0.0, "__g" : 0.0, "__p" : 0.0}

  S0 = Strategy('Algo0', '../Metasolver/BSG_CLP', '--alpha=__a -
beta=__b --gamma=__g -p __p -t 2 --min_fr=0.98', params)

  pi = range(800,1000)

  Sbest = bestParam(S0, "__a", [1.0, 4.0], pi, 0.00)


  Sbest = bestParam(Sbest, "__b", [1.0, 4.0], pi, 0.00)


  PI = '../Metasolver/extras/fw4exps/instancesBR.txt'

  sm = SpeculativeMonitor (experimentalDesign,PI)

  run()
```

Figura 6.1 Script diseño experimental

En el script podemos observar la definición de estrategias basado en sus atributos: nombre, ruta del ejecutable, argumentos y lista de parámetros. Luego la selección de un subconjunto de las instancias totales 200 de 1600 en total.

Luego se procede a la comparación, mediante el uso de la función **bestParam**, que recibe una estrategia, el parámetro a estudiar y una lista de valores para el parámetro. Esta función retorna en la variable **sbest** una copia de la estrategia **S0**, pero con el parámetro actualizado al mejor valor de la lista.

Una vez encontrado el mejor valor para el parámetro a, volvemos a ejecutar la función **bestParam**, pero esta vez buscando el mejor valor para el parámetro b.

En la figura (*Figura 6.2 Grafico calidad solución v/s n° instancias primera prueba*) podemos observar la calidad de solución aproximada para la ejecución del diseño de experimento con el *Framework*, en un servidor con un procesador de 16 cores. Se puede apreciar que la calidad de la solución entregada oscila al comienzo y más adelante converge valor óptimo

1, es decir tiene una certeza acerca del resultado de las comparaciones. Cada instancia se ejecutó con un tiempo de 30 segundos. se escogió un subconjunto de 200 instancias. Además, podemos apreciar la línea de tendencia que posee la curva de calidad, creciente y positiva.

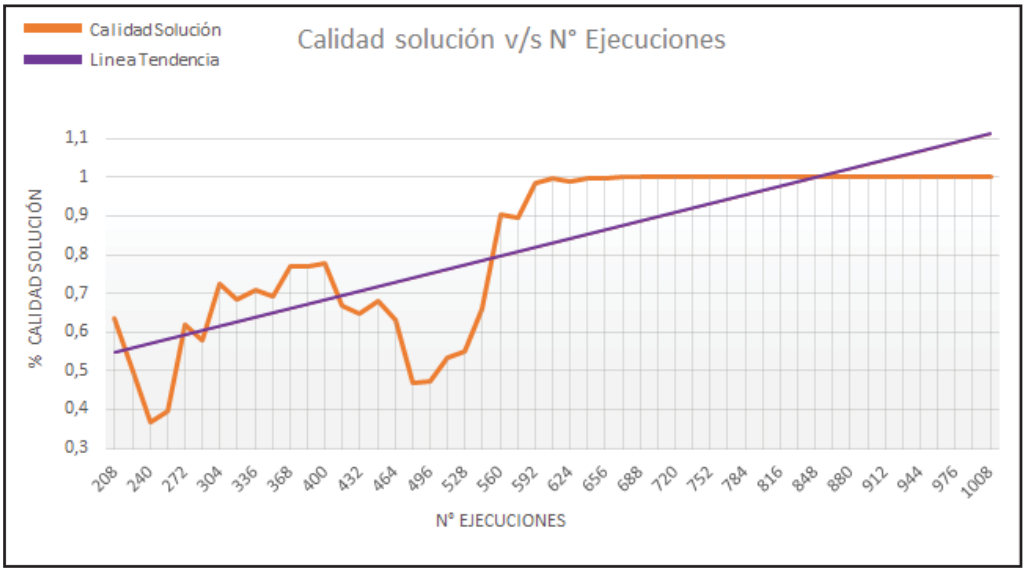

Figura 6.2 Grafico calidad solución v/s n° instancias primera prueba

En la (*Figura 6.3 Grafico calidad solución v/s n° instancias segunda prueba*) de manera similar podemos ver la ejecución del mismo diseño de experimentos en un subconjunto de 100 instancias, un procesador de 4 cores, y tiempo de 5 segundos por instancia. Aquí también se observa el crecimiento de la calidad en el tiempo convergiendo al valor óptimo de calidad.

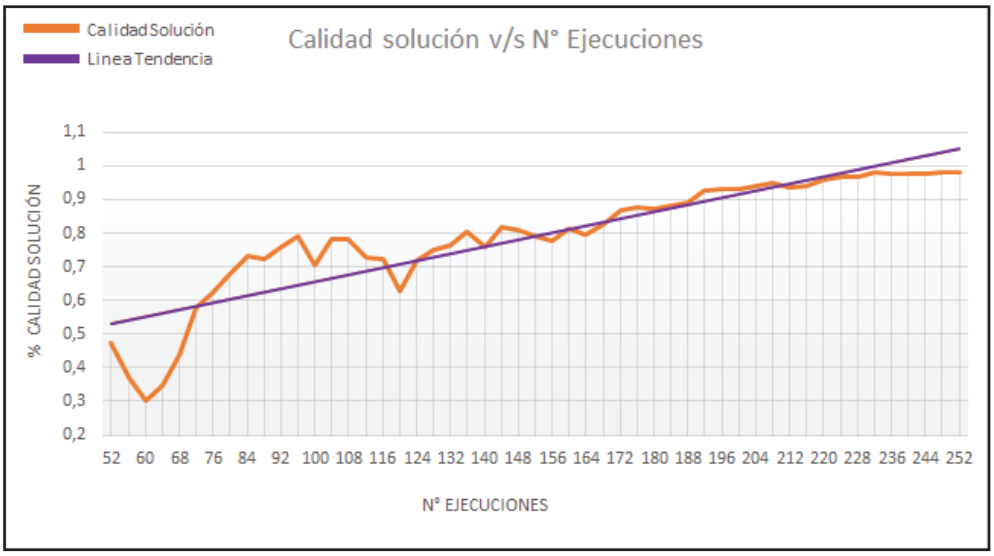

Figura 6.3 Grafico calidad solución v/s n° instancias segunda prueba

Respecto a ambos gráficos es posible afirmar que se pueden obtener conclusiones acerca de las comparaciones del diseño de experimentos con una calidad de certeza aceptable, alrededor del 80%, cerca del primer tercio de ejecuciones y una certeza total luego de ejecutar cerca del 50 % de las instancias, quedando demostrada la utilidad del *Framework*.

7. Conclusión

En esta entrega final de proyecto de titulo se consigue puntos relevantes para el funcionamiento actual del *Framework anytime para diseño de experimentos* respecto a sus versiones previas, ya que, este se encuentra en una fase operativa para el uso de cualquier investigador. Aunque es necesario realizar mejoras y actualización para seguir perfeccionando sus componentes principales y que el *Framework* pueda ser utilizado en cualquier plataforma.

Los últimos cambios realizados al *Framework* efectivamente maximizaron el incremento de la calidad de los resultados de las comparaciones durante el tiempo. En consecuencia, es posible afirmar que el método Monte Carlo nos acerco aun más a las características ideales a algoritmo *anytime* dado que la naturaleza de la calidad de los resultados se muestra menos errática y oscilante, es decir, más estable con sus versiones anteriores.

Los experimentos reflejaron que el *Framework anytime para diseño de experimentos* puede encontrar certezas alrededor del 80% de confianza una vez ejecutada un tercio de las instancias. Además, una certeza total de confianza una vez ejecutada la mitad de las instancias. Sumado a esto facilitamos el tiempo entrega de estos resultados, ya que se utiliza los recursos disponibles en la maquina mediante el uso de la paralelización automática.

En futuros cambios para el *Framework para diseño de experimentos* estarán orientados en la inclusión de utilidades y herramientas, tales como la pausa y reanudación del experimento, generación de archivos de resultados, tablas comparativas, la portabilidad, entre otras características según las necesidades de la comunidad investigadora.

8. Referencias

[1] Markus Chimani and Karstenkein, Algorithm Engineering: Concepts and Practice,Experimental Methods for the Analysis of Optimization algorithms, Springer 2010.

[2]J Silberholz, B Golden, Comparison of metaheuristics - Handbook of metaheuristics, 2010 – Springer

[3] Zilberstein, S. (1996). Using anytime algorithms in intelligent systems. AI magazine, 17(3), 73.

[4] Sobol′, I. M. (1976). Método de Montecarlo (No. Folleto 5124Y).

[5] Thomas Bartz-Beielstein and Mike Preuss, The Future of Experimental research, Experimental Methods for the Analysis of Optimization algorithms, Springer 2010.

[6] Bernd Bischl, Jakob Richter, Jakob Bossek , Daniel Horn, Janek Thomas Michel Lang, mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions,Computational Statistics & Data Analysis 2017

[7] Bartz-Beielstein, T., Gentile, L., & Zaefferer, M. (2017). In a Nutshell: Sequential Parameter Optimization.

[8] George, C. C. (1988). Probabilidad y estadística, aplicaciones y métodos. *McGrawHill/Interamericana de México, SA. México*

[9] Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. Journal of Machine learning research, 7(Jan), 1-30.

[10] Du Prel, J.-B., Röhrig, B., Hommel, G., & Blettner, M. (2010). Choosing Statistical Tests: Part 12 of a Series on Evaluation of Scientific Publications. *Deutsches Ärzteblatt International*, *107*(19), 343–348. http://doi.org/10.3238/arztebl.2010.0343

[11] Singer, S., & Nelder, J. (2009). Nelder-mead algorithm. Scholarpedia, 4(7), 2928.

[12] Thomas Wiecki 2010, MCMC sampling https://twiecki.github.io/blog/2015/11/10/mcmc-sampling/