Proyecto de Simulación e Inteligencia Artificial: Optimización y simulación multi-agente por eventos de un restaurante

Ariel González Gómez
Alex Samuel Bas Beovides

9 de octubre de 2024

1 Introducción

El proyecto se centra en la simulación y optimización de un restaurante, buscando mejorar la experiencia del cliente y la eficiencia operativa mediante técnicas avanzadas de simulación multi-agente y optimización. El objetivo principal es maximizar las propinas obtenidas, simulando diferentes escenarios durante una noche de servicio. Para ello, se implementó una simulación multi-agente basada en eventos, donde los agentes representaron a los clientes y a los meseros del restaurante.

La optimización se realizó utilizando el Algoritmo de Recocido Simulado (*Simulated Annealing - SA*) para explorar el espacio de búsqueda de configuraciones del restaurante, incluyendo la disposición de las mesas y los pesos asignados a las reglas de comportamiento de los agentes. Además, se empleó lógica difusa (*fuzzy logic*) para calcular el porcentaje de propina en función de la calidad del servicio. La metodología propuesta combinó simulación, inteligencia artificial y optimización para encontrar soluciones que mejoraron el rendimiento del restaurante en términos de satisfacción del cliente y eficiencia del servicio. Se implementó una aplicación visual para la prueba del sistema, con configuración de la simulación u optimización en lenguaje natural a través de Grandes Modelos del Lenguaje (*Large Language Models - LLM*). Por último, se realizaron experimentos y pruebas de hipótesis sobre el sistema desarrollado.

2 Simulación

El escenario simulado es una noche de trabajo de un restaurante. Este tiene forma rectangular y se representa como una matriz bidimensional con paredes en los bordes y consta de una sola entrada/salida, una cocina y una determinada cantidad de mesas y meseros. Hay una determinada cantidad de tiempo al inicio en la que el restaurante admite nuevas llegadas de clientes.

Por simplicidad, un cliente siempre sigue el mismo comportamiento: al llegar al restaurante, si no hay ninguna mesa libre empieza a esperar en la cola, en caso contrario es asignado a la mesa libre más cercana a la cocina por distancia euclideana, y empieza a caminar hacia esta; cuando llega a la mesa asignada, se sienta y empieza a pensar en qué ordenar; cuando decide su orden llaman al mesero más cercano a ellos por distancia euclideana, y empieza a esperar a ser atendido; cuando un mesero empieza a atenderlo, deja de esperar y le comienza a hacer la orden; cuando termina de ordenar, empieza a esperar a recibir su plato; cuando un mesero les entrega su plato pedido, deja de esperar y comienza a comer; cuando termina de comer, llama al mesero más cercano por distancia euclideana para que le recoja la cuenta, y empieza a esperar; cuando un mesero letrae la cuenta, deja de esperar por esta y empieza a pagarla; cuando termina de pagar la cuenta, empieza a caminar hacia la salida y se desocupa la mesa en la que estaba; cuando llega a la salida, se va del restaurante y, si habían clientes esperando a que una mesa se desocupase, pasa el primero en la cola.

El comportamiento de los meseros es más complejo y depende de las tareas que tenga asignadas en cada momento y un conjunto de reglas para la decisión de la tarea a realizar. Todos los meseros comienzan la simulación esperando en la cocina. Las tareas que pueden realizar los agentes son recoger una orden de un cliente, recoger un plato preparado de la cocina, llevar un plato a un cliente, recoger la cuenta de un cliente, y regresar a la cocina. Cuando los meseros pasan por la cocina siempre dejan todas las órdenes recogidas, y se actualiza su conocimiento sobre las órdenes preparadas que hay en la cocina.

La cocina tiene una capacidad limitada de platos que puede estar preparando a la vez, y siempre que tenga órdenes por preparar llena todas las capacidades que pueda o necesite.

Primeramente ocurre la configuración por parte del usuario. Esta se puede hacer modificando el archivo *config.json* o a través de la aplicación visual y la escritura en lenguaje natural a un *LLM*. Existen dos modos, una sola simulación, o la optimización del restaurante.

En esta configuración se define la cantidad de tiempo que el restaurante admite nuevas llegadas de clientes, la tasa de llegada de clientes, el restaurante a partir la cantidad de meseros y de un mapa del restaurante, matriz bidimensional de enteros que representan los tipos de celda: pared (0), piso libre (1), mesa (2), cocina (3), puerta de entrada/salida (4). Adicionalmente para el modo de optimización se pueden definir los hiperparámetros del Recocido Simulado (*Simulated Annealing - SA*) y la cantidad de simulaciones que agrega la función objetivo.

Tras la configuración por parte del usuario, inicia la simulación (u optimización). Como inicialización de cada simulación, se crea la cola de eventos, inicialmente vacía, se inicializa el tiempo actual (en $0s$) y un sistema precomputado de control de lógica difusa para determinar las propinas. A continuación, se generan eventos de llegada de clientes siguiendo un proceso de Poisson. Se calcula el tiempo de llegada de cada cliente (basado en una distribución exponencial con tasa *lambda_rate*) y se añade un evento de llegada de cliente a la cola de eventos.

La simulación se ejecuta procesando en orden cronológico los eventos de la cola de eventos. Para cada uno de estos, se actualiza el tiempo actual y se procesa el evento. Finalmente, se devuelve el total de propinas recibidas durante la simulación.

2.1 Suposiciones para simplificar la simulación

- Todos los clientes tienen el mismo plan de acción, y no se desvían del mismo, y todos los meseros tienen las mismas reglas que dirigen su curso de acción (todo esto explicado en esta sección).
- Todos los eventos y acciones que desencadenan otros eventos determinan, en el momento de generación, la duración del evento/acción actual y/o el tiempo exacto en el que ocurre el evento generado, y estos valores son inmutables.
- Un turno/iteración de la simulación equivale a exactamente un segundo de duración, y por tanto todas las acciones y eventos de la simulación duran un número entero exacto de segundos.
- El restaurante es una matriz rectangular perfecta, donde cada celda ocupa un espacio de exactamente un metro de ancho por un metro de alto (y por tanto las mesas, la entrada/salida, y la entrada a la cocina, tienen ese mismo tamaño).
- Todas las personas caminan a exactamente $1m/s$, y ocupan una sola celda.
- Se admite que varias personas puedan estar caminando en una misma celda a la vez.
- Las llegadas de clientes al restaurante siguen un proceso de Poisson.
- El camino de cualquier agente entre un par de lugares de interés (entrada/salida, mesas, cocina), es siempre el mismo (se precalcula), y sigue un camino mínimo que se determina con el Algoritmo de Búsqueda en Anchura (*Breadth First Search - BFS*).
- Los clientes llaman al mesero más cercano a ellos (con distancia euclidiana) para cualquier pedido (de tomar orden o traer la cuenta) y este mesero es el que lo cumple eventualmente.
- Tomar una orden toma un tiempo corto (entre 60 y 300 segundos) y al azar (distribución uniforme).
- Recibir un pago comienza desde que el mesero deja la cuenta y se queda en la mesa hasta terminar el pago, el cual toma un tiempo corto (entre 60 y 180 segundos) y al azar (distribución uniforme).

- Todos los platos que se sirven en el restaurante están destinados a ser servidos calientes, por lo que la temperatura ideal de estos es de alrededor de $70°C$, por lo que una temperatura muy inferior a esta causa disgusto a los clientes.
- Todos los platos tienen un tiempo promedio de cocinado, y el tiempo real que toma una orden en cocinarse se determina como este tiempo promedio más menos una cantidad de segundos al azar de entre 0 y 60 segundos (distribución uniforme). Además, la temperatura inicial de los platos al ser preparados se determina al azar en el rango de $70°C$ a $75°C$ (distribución uniforme).
- La cocina tiene una cantidad determinada de platos que puede preparar a la vez, y estos son los primeros de la cola de las órdenes que traen los meseros.
- Los clientes son sentados en la mesa disponible más cercana a la cocina.
- Los clientes deciden la cantidad de propina que dejar teniendo en cuenta la cantidad total de tiempo que esperaron por cualquier servicio, y la temperatura que tenía su comida al momento de ser servida, y en nada más.
- Un mesero puede llevar un máximo de 2 platos a la vez.
- La temperatura ambiente en todo el restaurante se mantiene en todo momento a exactamente $25°C$.
- La temperatura de los platos sigue exactamente la aproximación que brinda la *Fórmula de Enfriamiento de Newton*, con una contante de enfriamiento de 0.09.
- Muchas acciones tienen duración nula, como: la asignación o no de mesa a la llegada de un cliente (o la salida de otro cuando el restaurante estaba lleno y habían clientes esperando), los inicios y finales de caminatas y esperas de los agentes y clientes respectivamente, los llamados a los meseros, las asignaciones de tareas y las tomas de decisiones de los meseros, dejar órdenes, platos sucios y cuentas pagadas en la cocina, y recoger platos preparados de la cocina, por parte de los meseros, el aviso a los meseros de que hay un nuevo plato preparado en la cocina, la limpieza de la mesa que hace un mesero, etc.

2.2 Simulación basada en eventos

La simulación basada en eventos es un tipo de simulación que se centra en los eventos que ocurren en un sistema y cómo estos afectan el estado del sistema. En lugar de simular el paso del tiempo de manera continua, esta técnica avanza en el tiempo solo cuando ocurre un evento.

Funcionamiento:

1. **Definición de eventos**: Se define un conjunto de eventos que pueden ocurrir en el sistema, como la llegada de un cliente, la finalización de un servicio, la salida de un cliente, etc. Cada evento tiene un tiempo asociado que determina cuándo ocurrirá.
2. **Cola de eventos**: Se utiliza una cola de eventos para almacenar los eventos que ocurrirán en el futuro. La cola está ordenada por tiempo de ocurrencia, de modo que el evento más próximo en el tiempo se encuentra en la cabecera.
3. **Procesamiento de eventos**: La simulación avanza en el tiempo saltando de un evento a otro. Cuando ocurre un evento, se procesa y se actualiza el estado del sistema. El procesamiento del evento puede implicar:
    - Actualizar el tiempo actual de la simulación.
    - Cambiar el estado de las entidades del sistema (por ejemplo, un cliente pasa de estar esperando a estar siendo servido).
    - Generar nuevos eventos (por ejemplo, el evento de un cliente que finaliza su pago genera un evento de salida).
4. **Simulación continúa mientras la cola de eventos no esté vacía**: La simulación continúa procesando eventos hasta que no quedan eventos por procesar en la cola.

La simulación basada en eventos presenta muchas ventajas: es más eficiente que otras técnicas de simulación, ya que solo se calcula el estado del sistema cuando ocurre un evento; además, es muy flexible y se puede utilizar para simular una amplia variedad de sistemas; finalmente, puede ser muy precisa, especialmente si el sistema está bien definido. Todas estas ventajas determinaron el uso de este tipo de simulación en el proyecto, pues la eficiencia de una simulación individual es indispensable para garantizar la eficiencia final de todo el proceso de optimización, todo esto manteniendo un alto grado de flexibilidad y precisión.

2.3 Eventos

A continuación se describen los eventos que pueden ocurrir en la simulación del restaurante. Cada evento representa un cambio en el estado del sistema, desencadenando otros eventos.

- *CustomerArrives*: Representa la llegada de un cliente al restaurante. Si hay una mesa libre, el cliente empieza a caminar hacia ella, y se genera un evento *CustomerSits*. En caso contrario, el cliente espera al final de una cola hasta que sea el primero de esta y una mesa se desocupe (en algún evento *CustomerLeaves*).
- *CustomerSits*: Representa el momento en que un cliente termina de caminar hasta su mesa y se sienta. El cliente entonces comienza a pensar en su orden, generando un evento *CustomerOrders*.

- *CustomerOrders*: Representa el momento en que el cliente decide qué ordenar y llama al mesero más cercano a él por distancia euclideana. Este mesero recibe esta encomienda y cuando decida cumplirla se genera un evento *WaiterStartsTakingOrder*.

- *WaiterStartsTakingOrder*: Representa el momento en que un mesero llega a una mesa y comienza a tomar la orden de un cliente, el cual deja de esperar. Este evento genera un evento *WaiterTakesOrder*.

- *WaiterTakesOrder*: Representa el momento en que el mesero termina de tomar la orden del cliente, el cual empieza a esperar su plato.

- *WaiterDeliversDish*: Representa el momento en que el mesero llega a la mesa y entrega el plato ordenado al cliente. El cliente entonces deja de esperar y empieza a comer, generando un evento *CustomerFinishesEating*.

- *WaiterDeliversDish*: Representa el momento en que el mesero llega a la mesa y entrega el plato ordenado al cliente. El cliente entonces deja de esperar y empieza a comer, generando un evento *CustomerFinishesEating*.

- *CustomerFinishesEating*: Representa el momento en que el cliente termina de comer y llama a un mesero (el más cercano) para que traiga la cuenta. Este mesero recibe esta encomienda y cuando decida cumplirla se genera un evento *WaiterDeliversBill*.

- *WaiterDeliversBill*: Representa el momento en que el mesero llega a la mesa y entrega la cuenta al cliente. El cliente entonces deja de esperar y comienza a pagar la cuenta, generando un evento *CustomerPays*.

- *CustomerPays*: Representa el momento en que el cliente termina de pagar, deja una propina y se levanta de la mesa para dirigirse a la salida (generando el evento *CustomerLeaves*). El mesero, por su parte, limpia la mesa y empieza a regresar con los platos y la cuenta a la cocina (generando un evento *WaiterCleansTable*).

- *WaiterCleansTable*: Representa el momento en que el mesero regresa a la cocina para dejar los platos, la cuenta pagada, y las órdenes recogidas.

- *WaiterReturnsToKitchen*: Representa el momento en que el mesero regresa a la cocina. En esta el mesero deja las órdenes recogidas.

- *KitchenPreparesOrder*: Representa el momento en que la cocina termina de preparar una orden.

- *CustomerLeaves*: Representa el momento en que el cliente llega a la salida y se va del restaurante. Si habían clientes esperando por que se desocupara una mesa, entra el primero en la cola, el cual deja de esperar y comienza a caminar hacia la mesa desocupada, generando un evento *CustomerSits*.

2.4 Simulación multi-agente

La simulación multi-agente (SMA) es una técnica computacional que permite modelar sistemas complejos como entidades individuales, denominadas agentes, que interactúan entre sí y con el entorno. Cada agente posee su propio comportamiento y toma decisiones de forma autónoma, lo que permite a la SMA capturar la complejidad emergente de sistemas reales. Esta técnica se distingue de las simulaciones tradicionales, que suelen modelar el sistema como un todo, por su enfoque bottom-up, donde la interacción de agentes simples crea patrones y comportamiento a nivel macro.

En el contexto del proyecto, la SMA se utiliza para modelar el comportamiento de los clientes y meseros. La simulación del restaurante, como fue descrita, se basa en un enfoque de eventos discretos, donde el tiempo avanza de forma discontinua, y los eventos, como la llegada de un cliente o la solicitud de un plato, desencadenan acciones de los agentes. Esta estructura se asemeja a los autómatas celulares, donde la interacción entre agentes vecinos y con el entorno determina su estado en el tiempo.

La elección de la SMA para simular el restaurante se basa en la complejidad del sistema. Modelar el comportamiento individual de cada cliente, la interacción con los meseros, la dinámica de la cocina y la gestión de mesas requiere una perspectiva granular que la SMA ofrece. Cada agente, cliente o mesero, es una entidad independiente con sus propios objetivos y comportamiento, y las decisiones individuales de cada agente impactan en el comportamiento del sistema como un todo.

2.4.1 Agentes (*Agent*)

La clase *Agent* representa la base de la simulación multi-agente, encapsulando las características y comportamientos esenciales de cada entidad individual que participa en el sistema. Cada agente posee un identificador único, una posición actual, un camino actual, y un tipo que define su rol en la simulación. Además, la clase permite a los agentes moverse a través de rutas predefinidas. Además, se cuenta con una función para conocer la posición del agente en un momento específic, lo que permite sincronizar el movimiento de los agentes con el avance del tiempo de la simulación.

La clase *Agent* ofrece un marco flexible y adaptable para modelar una amplia gama de comportamientos. La capacidad de definir rutas de movimiento, controlar la posición en el tiempo y gestionar el inicio y finalización de los desplazamientos permite simular interacciones complejas entre los agentes. La modularidad de la clase facilita la integración de diferentes tipos de agentes, con características y comportamientos específicos, para crear sistemas de simulación multi-agente sofisticados y realistas.

2.4.2 Clientes (*Customer*)

La clase *Customer* extiende la clase *Agent* para modelar el comportamiento de los clientes en el restaurante. Además de las características generales de

los agentes, como el identificador, la posición y la capacidad de moverse, los clientes poseen atributos específicos: el tiempo total que el cliente ha esperado, el momento en que comenzó la espera actual, el índice de la mesa asignada, y la temperatura a la que fue servido su plato.

La clase *Customer* implementa funciones adicionales para simular las acciones de un cliente en el restaurante. Funciones como iniciar el conteo del tiempo de espera, detenerlo y acumular el tiempo total, recibir el plato ordenado, y registrar la propina dejada al finalizar la visita. Esta funcionalidad permite capturar la dinámica de la experiencia del cliente, incluyendo el tiempo de espera, la satisfacción y el comportamiento al finalizar la visita.

2.4.3 Meseros (*Waiter*)

La clase *Waiter* representa al mesero, un agente con un rol complejo que involucra diferentes tareas y la gestión de recursos. Además de los atributos generales de un *Agent*, como la posición e identidad, el mesero posee atributos específicos para modelar su trabajo: encomiendas pendientes de los clientes, órdenes recogidas, los platos listos para servir que está cargando el mesero actualmente, la cantidad de platos que puede llevar al mismo tiempo, la tarea actual que está llevando a cabo, y las prioridades de unas reglas de decisión.

La clase *Waiter* implementa funciones para gestionar las tareas del mesero: añadir una nueva encomienda a la lista, decidir la próxima tarea a atender, trabajar en una de las tareas asignadas, terminar la tarea actual, recibir una nueva orden, añadir un plato preparadoa a la carga actual del mesero, conocer si el mesero está disponible (no tiene ninguna tarea actual ni restante), y verificar si tiene capacidad para llevar más platos. Estas funciones permiten simular el flujo de trabajo del mesero, desde la recepción de solicitudes hasta la entrega de platos a los clientes.

2.4.4 Reglas de Decisión para Meseros

Se implementaron un conjunto de reglas de decisión que guían las acciones de los meseros. Estas reglas modelan el comportamiento complejo de un mesero en el restaurante, priorizando tareas y gestionando recursos de forma eficiente. Las reglas se evalúan cada vez que el mesero termina una tarea, y determinan la próxima acción del mesero.

Cada regla tiene asociado un peso, que define su prioridad en el orden de evaluación. Las reglas se evalúan secuencialmente según su peso, y aunque pueden cumplirse varias dependiendo de las reglas implementadas, actualmente la primera que se cumpla determina la siguiente acción del mesero. Esto permite que el sistema priorice tareas que determine urgentes, como la entrega de platos o la recolección de cuentas, sobre acciones menos prioritarias, como la búsqueda de nuevos platos en la cocina.

A continuación, se describe cada una de las reglas implementadas por defecto:

- *TakeFirstDish*: Si el mesero tiene espacio disponible, no tiene tarea actual, está en la cocina y hay al menos un plato preparado, el mesero toma el primer plato preparado disponible.
- *DeliverFirstDish*: Si el mesero tiene al menos un plato en su carga y no tiene tarea actual, entrega el primer plato que recogió.
- *CollectFirstBill*: Si el mesero no tiene tarea actual y tiene al menos una petición de cuenta por cobrar, cobra en la mesa que tenga una cuenta pendiente que hizo la primera petición.
- *TakeFirstOrder*: Si el mesero no tiene tarea actual y tiene al menos un pedido para tomar, toma el pedido pendiente de la mesa que hizo la primera petición.
- *GoToKitchen*: Si el mesero no tiene tarea actual y no está en la cocina, se dirige a la cocina.
- *DoFirstTask*: Si el mesero no tiene tarea actual y tiene al menos una tarea pendiente, realiza la primera tarea en la lista.

Este sistema de reglas define un comportamiento flexible y adaptable para los meseros en la simulación. Al priorizar las tareas urgentes y gestionar los recursos de forma eficiente, las reglas permiten modelar el comportamiento real de un mesero en un restaurante. La permutación de los pesos de estas reglas durante la etapa de optimización permiten al sistema aprender la mejor estrategia de los meseros, para hacer más eficaz el servicio y aumentar las propinas obtenidas.

2.4.5 Clasificación y descripción de los agentes

Los agentes en la SMA del restaurante constituyen *agentes reactivos basados en modelo*, con un enfoque en el comportamiento observable, y reglas de condición-acción. Los agentes (clientes y meseros) reaccionan a eventos del entorno (llegada de clientes, pedidos, etc.) y ajustan su comportamiento en consecuencia. Aunque no se implementa explícitamente el uso de estados del modelo *BDI (Beliefs-Desires-Intentions)*, la simulación presenta elementos de creencias (por ejemplo, el mesero tiene una *"belief"* sobre la disponibilidad de platos), deseos (los clientes *"desean"* ser atendidos, los meseros *"desean"* cumplir las tareas asignadas) e intenciones (el mesero *"itenta"* completar las solicitudes). Estas creencias y deseos se reflejan en los atributos de las clases y las funciones que ejecutan.

Aunque los agentes no tienen una lógica de aprendizaje explícita, las acciones se basan en un modelo de comportamiento predefinido. Por ejemplo, la clase *Customer* utiliza funciones predefinidas para modelar la experiencia del cliente. Este modelo, basado en reglas y heurísticas, define el comportamiento del agente ante diferentes situaciones. Es decir, las acciones no son simplemente reacciones directas a estímulos, sino que se basan en una comprensión del contexto (tiempo de espera, disponibilidad de mesas) y un conjunto de reglas que

guían la respuesta del agente. Además, el aprendizaje de los agentes meseros es posible a través de la permutación de los pesos de sus reglas de decisión, la modificación o eliminación de algunas, o la inclusión de reglas nuevas, más o menos complejas.

2.5 Cálculo de la propina: Lógica difusa (*Fuzzy Logic*)

Para simular el comportamiento de un cliente al determinar el porcentaje de propina, fue desarrollado un sistema de lógica difusa *fuzzy logic* que considera las variables de tiempo de espera y temperatura de la comida. El sistema, implementado en Python utilizando la biblioteca *Scikit-fuzzy* (*skfuzzy*), define conjuntos difusos para cada variable de entrada y salida. Los conjuntos difusos para el tiempo de espera (*short*, *acceptable*, *long*, ver figura 1) medido en segundos (s) y la temperatura de la comida (*cold*, *hot*, ver figura 2) medida en grados Celsius ($°C$) representan el grado de pertenencia de un valor específico a cada término lingüístico. La salida, el porcentaje de propina (*low*, *medium*, *high*, ver figura 3), refleja el nivel de satisfacción del cliente.

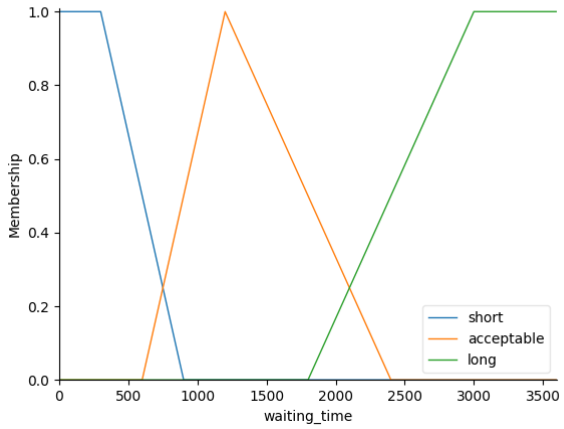

Figure 1: Función de pertenencia de Tiempo de Espera.

Las reglas difusas, que relacionan las variables de entrada con la salida, se definen utilizando operadores lógicos como *AND* ($\wedge$), *OR* ($\vee$) e *implicación* ($\implies$).

$$R_1 = ([W_{short} \wedge T_{hot}] \implies P_{high})$$
$$R_2 = ([W_{acceptable} \vee T_{cold}] \implies P_{medium})$$
$$R_3 = (W_{long} \implies P_{low})$$

La primera regla establece que una propina alta es más probable si el tiempo de espera es corto y la comida fue servida caliente. La segunda regla sugiere una

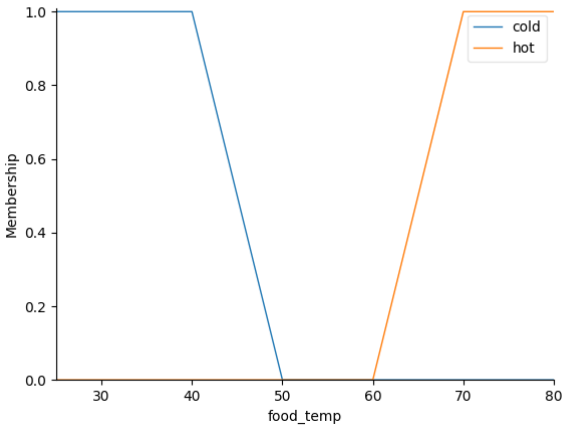

Figure 2: Función de pertenencia de Temperatura de la Comida.

propina mediana si el tiempo de espera es aceptable (ni muy corto ni muy largo) o la comida fue servida fría. Por último, la tercera regla indica una propina baja si el tiempo de espera es largo.

El sistema de lógica difusa, definido por las reglas y los conjuntos difusos, se utiliza inicialmente para precomputar el porciento de propina más probable para todas las posibles entradas enteras de tiempo de espera y temperatura. El tiempo de espera, por las suposiciones para simplificar la simulación, es un número entero de segundos mayor o igual a 0, y el conjunto difuso de tiempo de espera va de $0s$ a $3600s$, donde este último valor pertenece únicamente a un tiempo de espera largo (1 para *long*, 0 para *short* y *acceptable*), por lo que valores de espera mayores a $3600s$ son igualados a este valor. Los platos preparados tienen temperatura inicial cercana a $80°C$, con temperatura ambiente de $25°C$, por lo que la temperatura del plato servido será un número real en este rango. Como es imposible precomputar todos los valores reales en un rango, se toman solo los valores enteros en este, y el valor de entrada es redondeado por defecto (ver figura 4).

Esta implementación de la lógica difusa para el Problema de la Propina proporciona una aproximación más realista a la complejidad del proceso de decisión del cliente, considerando las características subjetivas y el grado de satisfacción.

2.5.1 Determinación de la Temperatura del Plato: Aplicación de la Fórmula de Enfriamiento de Newton

En el contexto del problema de la propina, la temperatura del plato servido juega un papel crucial en la percepción del cliente sobre la calidad del servicio. Para determinar la temperatura del plato al momento de ser servido, se implementó

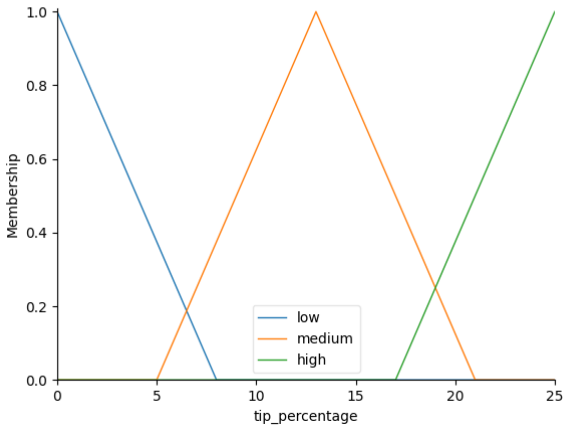

Figure 3: Función de pertenencia de Porciento de Propina.

la Fórmula de Enfriamiento de Newton. Esta fórmula, basada en principios físicos, permite calcular la temperatura de un objeto en función del tiempo transcurrido, la temperatura del ambiente y la constante de enfriamiento.

La fórmula de enfriamiento de Newton se expresa de la siguiente manera:

$$T(t) = T_a + (T_0 - T_a)e^{-kt}$$

Donde:

- $T(t)$: Temperatura del plato en el tiempo t.
- T_a: Temperatura del ambiente ($25°C$).
- T_0: Temperatura inicial del plato (entre $70°C$ y $75°C$).
- k: Constante de enfriamiento (0.09).
- t: Tiempo transcurrido entre la preparación del plato y el momento de servirlo.

Parámetros utilizados:

- Temperatura Inicial (T_0): La temperatura inicial del plato se considera entre $70°C$ y $75°C$, reflejando el rango típico de temperatura de los platos al ser preparados en la cocina.
- Temperatura del Ambiente (T_a): Se mantiene constante a $25°C$, asumiendo un ambiente de servicio normal.
- Constante de Enfriamiento (k): Se fija en 0.09, representando el ritmo de enfriamiento del plato en el ambiente dado.

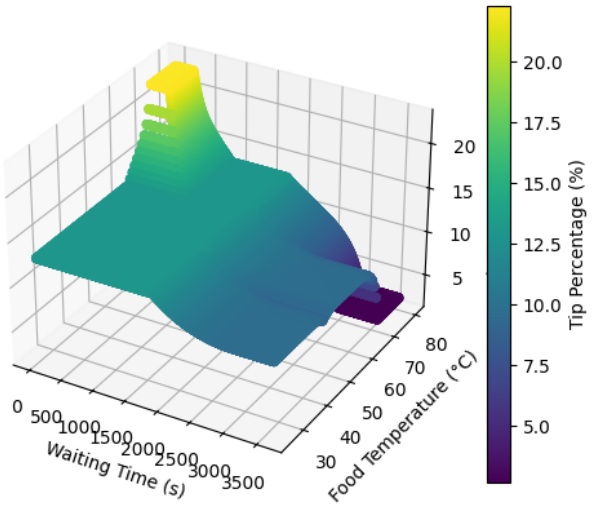

Figure 4: Función 3D de Tiempo de Espera y Temperatura de la Comida a Porciento de Propina.

Es importante mencionar que la Fórmula de Enfriamiento de Newton, al ser un modelo simplificado, no contempla la influencia de factores adicionales como el tamaño del plato, la forma del contenedor o la interacción con otros elementos en el ambiente. Sin embargo, para los fines de este sistema de lógica difusa, la fórmula ofrece una aproximación razonablemente precisa de la temperatura del plato, lo que permite una mejor estimación de la propina considerando la satisfacción del cliente.

3 Optimización. Recocido Simulado (*Simulated Annealing - SA*)

El proyecto de optimización del restaurante, orientado a maximizar las propinas recibidas, se basa en la exploración del espacio de búsqueda del diseño y manejo del restaurante mediante un algoritmo de Recocido Simulado (*Simulated Annealing - SA*). Esta metaheurística, inspirada en la física de la metalurgia, permite la búsqueda eficiente de una solución óptima dentro de un espacio de búsqueda complejo y de alta dimensionalidad.

El *SA* opera iterativamente sobre el espacio de búsqueda, explorando distintas configuraciones del restaurante. Cada configuración se define a través de un conjunto de parámetros, como la distribución de las mesas, el número de

meseros, la capacidad de la cocina, entre otros. La función de costo, definida como el promedio de la suma de las propinas recibidas en un número determinado de simulaciones, se evalúa para cada configuración.

El algoritmo se basa en la simulación de un proceso de enfriamiento lento de un material metálico, permitiendo que el sistema explore el espacio de búsqueda de forma aleatoria, con una probabilidad de aceptar configuraciones de menor calidad que aumenta conforme el "enfriamiento" progresa. Este proceso permite escapar de mínimos locales y alcanzar una solución cercana al óptimo global.

El *SA* ofrece una serie de ventajas para la optimización del restaurante:

- **Exploración del Espacio de Búsqueda**: El algoritmo permite una exploración exhaustiva del espacio de búsqueda, evitando quedar atrapado en mínimos locales.
- **Tolerancia al Ruido**: La naturaleza estocástica del algoritmo permite lidiar con la incertidumbre y el ruido inherente en las simulaciones.
- **Optimización Continua**: El algoritmo se puede utilizar para optimizar el diseño y manejo del restaurante de forma continua, adaptándose a cambios en las condiciones del entorno.

El uso de esta metaheurística garantiza una exploración eficiente del espacio de búsqueda, contribuyendo a la obtención de una solución óptima para maximizar las propinas recibidas en el restaurante.

3.1 Función objetivo y Espacio de búsqueda

En este contexto, el *SA* se aplica para determinar el mejor diseño y manejo del restaurante, utilizando la configuración inicial generada por los Grandes Modelos del Lenguaje (*Large Language Models - LLM*). Como función objetivo, se utiliza el promedio de la suma de las propinas recibidas en un número determinado de simulaciones, ya que esto constituye un buen indicador de la eficiencia de la distribución de recursos y el servicio. Se realizaron experimentos que demostraron que aproximadamente 10 simulaciones por configuración son suficientes para que la función de costo converja.

El espacio de búsqueda se constituye de dos componentes principales:

1. **Disposición de las Mesas**: Se representa mediante una matriz bidimensional de enteros que representan los tipos de celda: pared (0), piso libre (1), mesa (2), cocina (3), puerta de entrada/salida (4). La disposición válida debe cumplir con ciertas restricciones, como la presencia de paredes en el borde de la matriz, la ubicación única de la cocina y la puerta, la cantidad correcta de mesas y la no adyacencia de las mesas entre sí, ni a la cocina o la puerta. La inclusión de esta disposición en el espacio de búsqueda persigue la optimización de un aspecto que puede ser de importancia en la rapidez del servicio de un restaurante, y cuya implementación en el mundo real tiene un costo ínfimo (mover las mesas).

2. **Pesos de las Reglas de Decisión de los Meseros**: Como se expuso en secciones anteriores, los meseros cuentan con un conjunto de reglas que determinan la forma en que deciden la siguiente acción a realizar. Estas reglas tienen un peso o prioridad asignados, y se evalúan siguiendo este orden. Se representan mediante una permutación de los pesos, como un arreglo de enteros. Cada peso define la prioridad de una regla específica que determina la acción a realizar por el mesero. Optimizar la permutación de prioridades se traduce en un aprendizaje de los meseros a tomar mejores decisiones basadas en el contexto y la situación, a través de la experiencia.

En el *SA*, se parte de una configuración inicial, y cada configuración de entrada es sucedida por una configuración vecina. En el proyecto una configuración vecina se obtiene intercambiando una celda de piso libre por una celda de mesa, y/o de los pesos de dos reglas distintas, lo que implica solo revisar que se cumpla la condición de no adyacencia a otra mesa, la cocina o la puerta.

3.1.1 Analogía con el Aprendizaje Real

La aplicación del algoritmo *SA* al restaurante se puede analogar con el aprendizaje real de la dirección del restaurante y los meseros a través de la experiencia:

1. **Disposición de las Mesas**: Análogo a la experimentación de la dirección del restaurante con diferentes disposiciones de las mesas para optimizar el flujo de clientes y la eficiencia del servicio. El *SA* simula este proceso de prueba y error al explorar diferentes configuraciones de la disposición de las mesas y evaluar su impacto en las propinas recibidas.

2. **Pesos de las Reglas de Decisión**: Análogo al aprendizaje de los meseros a través de la experiencia, observando las acciones de los clientes y las respuestas del restaurante. El *SA* simula este aprendizaje al ajustar los pesos de las reglas de decisión de los meseros y observando su impacto en las propinas recibidas.

3.2 Configuración inicial de la simulación u optimización: Procesamiento de lenguaje natural (*Natural Language Proccessing - NLP*)

Se hizo uso de grandes modelos del lenguaje (*Large Language Models - LLM*) para el procesamiento de consultas en lenguaje natural del usuario para determinar la configuración inicial de la simulación (generación de un archivo *JSON* con una estructura predefinida).

Específicamente, se utilizó la *API* de *Gemini*, con un *prompt* que especifica el formato y la información a extraer, con sus llaves en el formato *JSON*, así como instrucciones para ignorar datos inexistentes o irrelevantes.