Proyecto de Simulación e Inteligencia Artificial [MVP]

Optimización y simulación multi-agente por eventos de un restaurante

Integrantes:

- Alex Samuel Bas Beovides
- Ariel González Gómez

Introducción

El proyecto consiste en la optimización de un restaurante, con el objetivo de obtener la mayor suma de propinas en promedio de varias simulaciones de una noche de trabajo en el restaurante. Se aplican componentes de Inteligencia Artificial, de búsqueda, conocimiento y procesamiento de lenguaje natural. Se utilizan 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). Se aplican metaheurísticas para la optimización, sobre el espacio de búsqueda del diseño y manejo del restaurante, cuya función de costo es la agregación de los resultados devueltos por las simulaciones. Se realizan simulaciones multi-agente, por eventos que representan las acciones de los agentes y la cocina en el restaurante. En estas se aplica conocimiento experto y heurísticas, y lógica difusa (fuzzy logic) para determinar el porciento de propina que deja el cliente.

Componentes de Inteligencia Artificial

Búsqueda

Metaheurísticas

Para determinar el mejor diseño y manejo del restaurante con la configuración dada, se explora el espacio de búsqueda mediante una metaheurística conocida como Algoritmo de Recocido (o Temple) Simulado (Simulated Annealing).

Camino mínimo en matriz bidimensional

Los caminos mínimos entre pares específicos de celdas de la matriz bidimensional que representa al restaurante, se calculan usando el Algoritmo de Búsqueda en anchura (Breadth First Search (BFS)).

Conocimiento

Lógica difusa (fuzzy logic)

La lógica difusa se aplica para determinar el porciento de propina que dejan los clientes, basándose en el tiempo total que esperaron en el restaurante (por una mesa, por que le tomen la orden, para recibir la comida y la cuenta).

Conocimiento experto

- Funcionamiento de un restaurante.
- Heurísticas para las decisiones de los meseros y la cocina.
- Movimiento de las personas (tomando caminos mínimos).
- Heurísticas para la decisión de la propina.
- Distribución de Poisson para la probabilidad de llegada de las personas al restaurante.
- Fórmula de enfriamiento de Newton para la determinación de la temperatura de un plato al momento de servirse.

Procesamiento de lenguaje natural (Natural Language Proccessing (NLP))

Se hace 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 JSON con una estructura predefinida).

Optimización. Simulated Annealing

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. 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 algoritmo de recocido simulado 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 la suma de las propinas recibidas en las 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 algoritmo de recocido simulado 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.

En este contexto, el algoritmo de recocido simulado se aplica para determinar el mejor diseño y manejo del restaurante, utilizando la configuración inicial generada por los Grandes Modelos del Lenguaje (LLM). Se ajustan iterativamente parámetros como la distribución de mesas, la asignación de meseros y la capacidad de la cocina, evaluando su impacto en la función de costo.

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.

Simulación

Tras la configuración por parte del usuario, se inicializa la simulación del restaurante: se define la cantidad de tiempo que el restaurante admite nuevas llegadas de clientes (los experimentos que se han realizado han sido con una hora, es decir, 3600 segundos), la tasa de llegada de clientes, y si se desea un modo `verbose`, para imprimir los eventos y acciones que van ocurriendo en la simulación. Se crea un objeto `Restaurant` 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). Se crea la cola de eventos, inicialmente vacía, se inicializa el tiempo actual (en 0s) y se inicializa un objeto de sistema 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.

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 (todos estos explicados en la sección de Eventos de este documento).
- 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.
- Los meseros realizan los pedidos en el orden que les fueron dados.
- 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 tienen un tiempo promedio de cocina, 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).
- 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 grados Celsius.
- La temperatura de los platos sigue exactamente la aproximación que brinda la fórmula de enfriamiento de Newton, con una contante de enfriamiento de 0.05.
- 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.

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.

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, genera el evento CustomerSits. En caso contrario, el cliente espera hasta que 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 a un mesero.
- `WaiterStartsTakingOrder`: Representa el momento en que un mesero llega a una mesa y comienza a tomar la orden de un cliente. Este evento genera un evento WaiterTakesOrder.
- `WaiterTakesOrder`: Representa el momento en que el mesero termina de tomar la orden del cliente.
- `WaiterDeliversDish`: Representa el momento en que el mesero llega a la mesa y entrega el plato ordenado al cliente. El cliente entonces comienza a comer, generando un evento CustomerFinishesEating.
- `CustomerFinishesEating`: Representa el momento en que el cliente termina de comer y llama a un mesero para que traiga la cuenta.
- `WaiterDeliversBill`: Representa el momento en que el mesero llega a la mesa y entrega la cuenta al cliente. El cliente entonces comienza a pagar, 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 regresa con los platos y la cuenta a la cocina (generando un evento WaiterCleansTable).
- `WaiterCleansTable`: Representa el momento en que el mesero termina de limpiar la mesa y comienza a regresar a la cocina para dejar los platos y la cuenta pagada.

Este evento genera un evento WaiterReturnsToKitchen.

- `WaiterReturnsToKitchen`: Representa el momento en que el mesero regresa a la cocina.
- `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.

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 ámbito de la simulación, la SMA encuentra aplicaciones en diversos campos, incluyendo la economía, la logística y la biología. En el contexto del proyecto, se simula un restaurante, la SMA se utiliza para modelar el flujo de clientes, el movimiento de meseros y la gestión de recursos. 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 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.

La simulación multi-agente del restaurante proporciona una herramienta poderosa para analizar y optimizar el funcionamiento del establecimiento. Al variar parámetros como la

cantidad de clientes, la velocidad de los meseros o la capacidad de la cocina, se pueden observar los efectos en la eficiencia del servicio, la satisfacción del cliente y la rentabilidad. Esta información puede utilizarse para identificar cuellos de botella, optimizar la distribución de recursos y mejorar la experiencia del cliente.

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. 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 (Belief-Desire-Intention), la simulación presenta elementos de creencias (por ejemplo, el mesero tiene una "belief" sobre la disponibilidad de platos), deseos (los clientes "desean" ser atendidos) e intenciones (el mesero "intenta" 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 como "start_waiting" y "leave_tip" 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.

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 (`id`), una posición actual (`position`) y un tipo (`type`) que define su rol en la simulación. Además, la clase permite a los agentes moverse a través de rutas definidas, implementando la función `start_walking` para iniciar un recorrido y `stop_walking` para finalizarlo. La posición del agente en un momento específico se obtiene utilizando la función `get_position_at` , 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.

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: `total_wating_time` registra el tiempo total que el cliente ha esperado, `cur_waiting_start_time` indica el momento en que comenzó la espera actual, `table_id` identifica la mesa asignada al cliente.

La clase `Customer` implementa funciones adicionales para simular las acciones de un cliente en el restaurante. `start_waiting` inicia el conteo del tiempo de espera, `stop_waiting` lo detiene y acumula el tiempo total, `leave_tip` permite 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.

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: `requests_queue` almacena las encomiendas pendientes de los clientes, `orders_queue` guarda las ordenes recibidas, `dishes` contiene los platos listos para servir que está cargando el mesero actualmente, y `capacity` define la cantidad de platos que puede llevar al mismo tiempo.

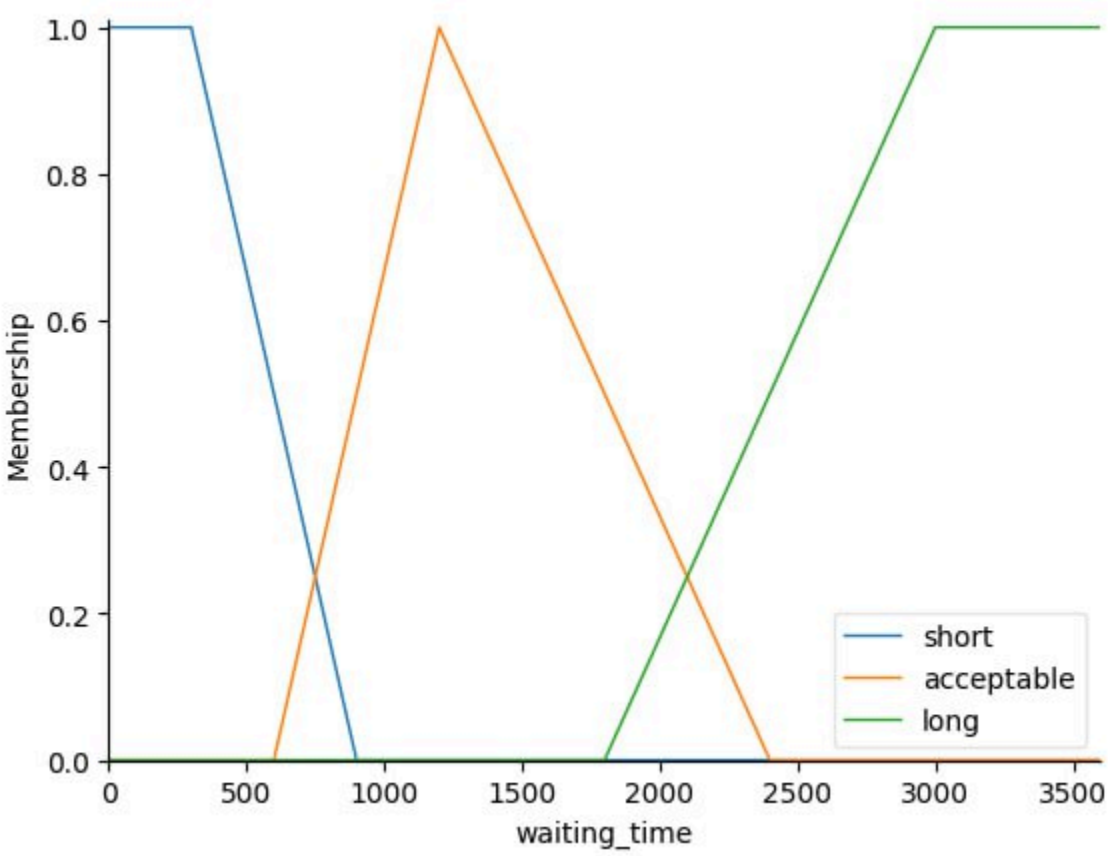
La clase `Waiter` implementa funciones para gestionar las tareas del mesero: `add_request` añade una nueva encomienda a la cola, `next_request` retorna la próxima encomienda a atender, `finish_next_request` marca la encomienda como completada, `add_order` añade una nueva orden, `add_dish` recibe un plato preparado y lo añade a la carga actual del mesero, `is_free` indica si el mesero está disponible (no tiene ninguna encomienda) y `has_capacity` verifica si tiene capacidad para llevar más platos. Estas funciones permiten simular el flujo de

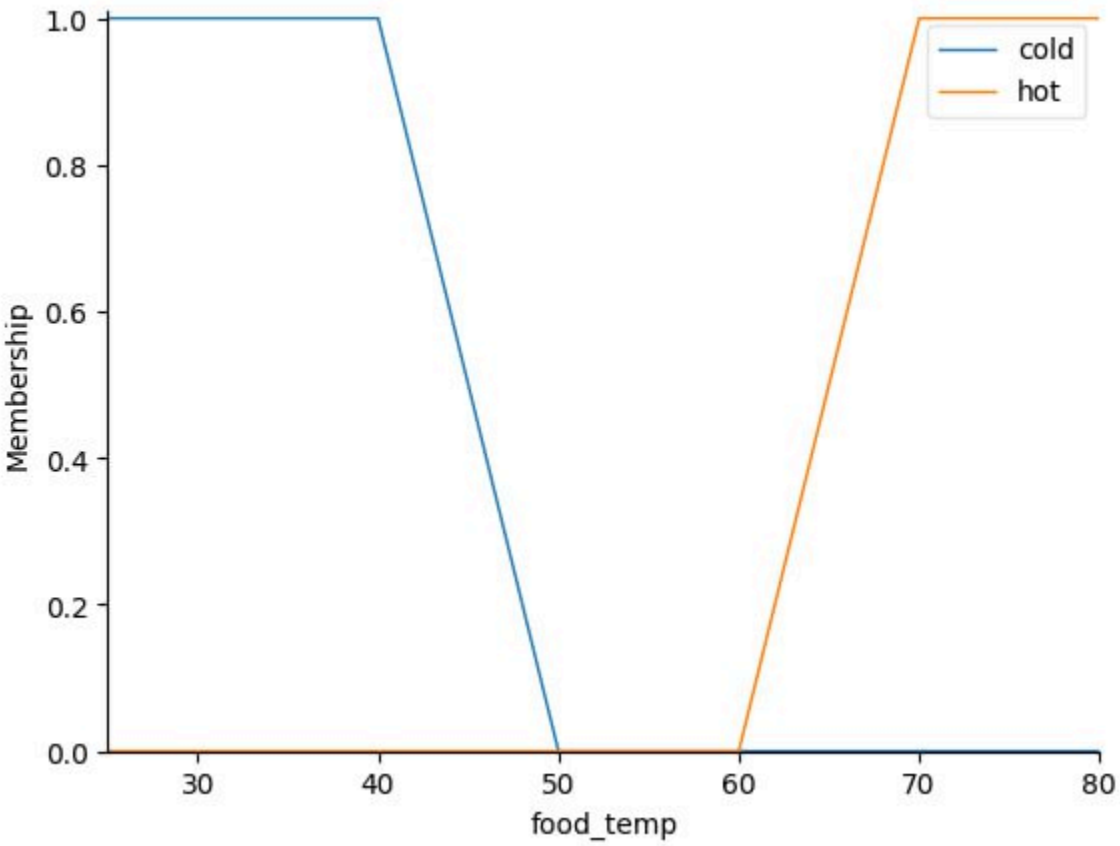
trabajo del mesero, desde la recepción de solicitudes hasta la entrega de platos a los clientes.

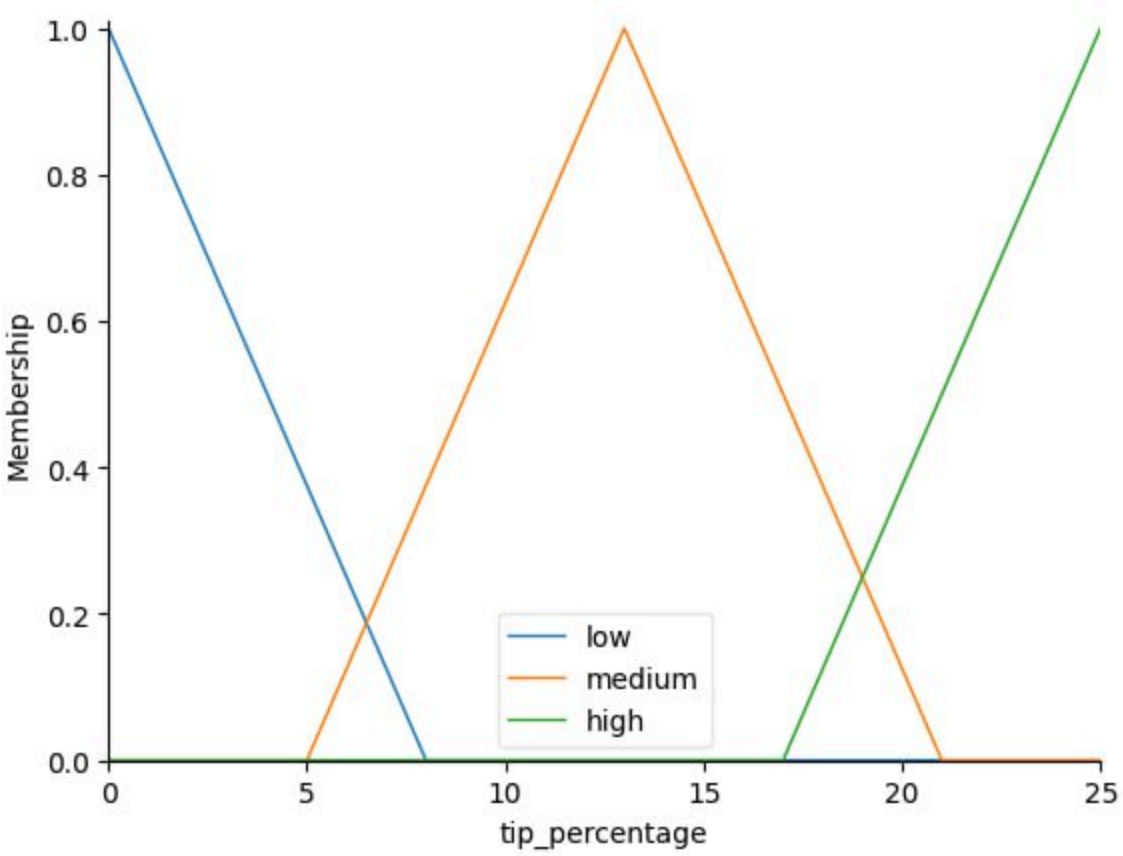
Lógica Difusa (Fuzzy Logic)

La lógica difusa, un enfoque de computación que utiliza conjuntos difusos para representar conceptos imprecisos, ha encontrado aplicaciones en una variedad de campos, incluyendo el control de procesos, la toma de decisiones y la modelación de sistemas complejos. En este trabajo, exploramos el uso de la lógica difusa para abordar el clásico Problema de la Propina, un problema de toma de decisiones que involucra factores subjetivos y multidimensionales.

Para simular el comportamiento de un cliente al determinar el porcentaje de propina, fue desarrollado un sistema de lógica difusa que considera las variables de tiempo de espera y temperatura de la comida. El sistema, implementado en Python utilizando la biblioteca Scikit-fuzzy, define conjuntos difusos para cada variable de entrada y salida. Los conjuntos difusos para el tiempo de espera ("short", "acceptable", "long") y la temperatura de la comida ("cold", "hot") representan el grado de pertenencia de un valor específico a cada término lingüístico. La salida, el porcentaje de propina ("low", "medium", "high"), refleja el nivel de satisfacción del cliente.




1.0
0.8
0.6
0.4
0.2
0.0
Membership
30
40
50
60
70
80
food_temp
cold
hot



Las reglas difusas, que relacionan las variables de entrada con la salida, se definen utilizando operadores lógicos como "AND", "OR" e "implicación".

```
rule1 = ctrl.Rule(self.waiting_time['short'] & self.food_temp['hot'], self.tip_percentage['high'])

rule2 = ctrl.Rule(self.waiting_time['acceptable'] | self.food_temp['cold'], self.tip_percentage['m

rule3 = ctrl.Rule(self.waiting_time['long'], self.tip_percentage['low'])
```

En el código presentado, la regla "rule1" establece que una propina "high" es más probable si el tiempo de espera es "short" y la temperatura de la comida es "hot", mientras que "rule2" sugiere una propina "medium" si el tiempo de espera es "acceptable" o la comida es "cold". Por último, "rule3" indica una propina "low" si el tiempo de espera es "long".

El sistema de lógica difusa, definido por las reglas y los conjuntos difusos, se implementa en la clase "FuzzyTip" del código. La función "get_tip" calcula el porcentaje de propina basado en los valores de entrada del tiempo de espera y la temperatura de la comida.

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. Se utilizó la biblioteca de Python Scikit-fuzzy (skfuzzy).

La biblioteca Scikit-fuzzy (skfuzzy) en Python es una herramienta fundamental para el desarrollo de sistemas de lógica difusa. Proporciona una interfaz fácil de usar para definir conjuntos difusos, reglas difusas y sistemas de control difuso. Esta biblioteca es esencial para modelar conceptos imprecisos y tomar decisiones basadas en información subjetiva, como en el Problema de la Propina. Skfuzzy permite la creación de conjuntos difusos con diferentes funciones de pertenencia (triangulares, trapezoidales, etc.), la definición de reglas usando operadores lógicos y la simulación de sistemas difusos para obtener salidas a partir de entradas. Esto facilita la creación de modelos que reflejan la complejidad del comportamiento humano y permiten la construcción de sistemas inteligentes que se adaptan a situaciones cambiantes. Todas estas características la vuelven una herramienta ideal para el proyecto presentado.

Autómatas celulares

Aunque el proyecto de optimización del restaurante se basa principalmente en la simulación multi-agente (SMA), la estructura de la simulación basada en eventos discretos presenta similitudes con el concepto de autómatas celulares. Estos sistemas computacionales, compuestos por una red de celdas con estados discretos que evolucionan en el tiempo según reglas simples, ofrecen un marco conceptual interesante para analizar la dinámica del restaurante simulado.

La analogía con los autómatas celulares radica en la posibilidad de representar elementos del restaurante como celdas con estados específicos. Por ejemplo, las mesas podrían ser celdas con estados "libre" u "ocupada", mientras que los meseros podrían representarse como celdas con estados "disponible" o "atendiendo". Las reglas de transición de los autómatas celulares, que determinan el estado futuro de cada celda en función de sus vecinos, podrían emular la interacción entre los agentes del sistema, como la llegada de un cliente que cambia el estado de una mesa de "libre" a "ocupada".

Si bien el proyecto actual no implementa autómatas celulares, la analogía con este enfoque ofrece una perspectiva complementaria para comprender la dinámica del restaurante. La aplicación de autómatas celulares como herramienta de análisis y visualización podría permitir

identificar patrones emergentes en el sistema, como la propagación de la ocupación de mesas o la dinámica de los meseros.

Resultados

Los experimentos con diversas configuraciones del restaurante, realizados durante el proceso de optimización, arrojaron resultados interesantes y coherentes con la intuición. La metaheurística de recocido simulado, al explorar el espacio de búsqueda de diseños y manejo del restaurante, encontró que la cercanía de las mesas a la cocina es un factor crucial para maximizar la ganancia en propinas.

Las simulaciones demostraron que la reducción del tiempo de espera para los pedidos, resultado de una disposición de las mesas más cercana a la cocina, incrementa la satisfacción del cliente y, por ende, la propina. El algoritmo de recocido simulado, al evaluar diferentes configuraciones, favoreció diseños que minimizaban la distancia entre las mesas y la cocina, siempre y cuando se mantuviera un flujo de circulación óptimo para los meseros.

En resumen, la optimización del restaurante, a través de la combinación de técnicas de IA y simulación, no solo encontró soluciones óptimas en términos de propinas, sino que también arrojó resultados que concuerdan con las mejores prácticas del diseño de restaurantes en el mundo real. La cercanía de las mesas a la cocina, al reducir los tiempos de espera, se reveló como un factor crítico para la satisfacción del cliente, lo cual se traduce en mayores propinas.

Conclusiones

La investigación ha demostrado la viabilidad de la optimización de un restaurante mediante la combinación de técnicas de Inteligencia Artificial y simulación multi-agente. El uso de Grandes Modelos del Lenguaje (LLM) para la configuración inicial de la simulación, junto con el algoritmo de recocido simulado para la exploración del espacio de búsqueda, ha permitido identificar configuraciones óptimas para maximizar las propinas recibidas. La simulación basada en eventos, con su enfoque granular en la interacción de agentes, ha proporcionado un modelo realista del funcionamiento del restaurante, permitiendo la evaluación de distintos escenarios y la identificación de puntos críticos.

La implementación de la lógica difusa para el Problema de la Propina ha añadido un nivel de realismo a la simulación, permitiendo modelar las decisiones de los clientes en función de factores subjetivos y el grado de satisfacción. Los resultados obtenidos sugieren que la aplicación de estas técnicas puede generar un impacto positivo en la rentabilidad del restaurante, tanto en términos de propinas como de satisfacción del cliente.

Los experimentos con distintas configuraciones de restaurante durante la optimización revelaron que la cercanía de las mesas a la cocina juega un papel crucial en maximizar las ganancias en propinas. La metaheurística de recocido simulado, al explorar el espacio de búsqueda de diseños y gestión, descubrió que la reducción de los tiempos de espera, resultado de una disposición de las mesas más cercana a la cocina, incrementa la satisfacción del cliente y, en consecuencia, la propina. El algoritmo, al evaluar diferentes configuraciones, favoreció diseños que minimizaban la distancia entre las mesas y la cocina, siempre y cuando se mantuviera un flujo de circulación óptimo para los meseros. Estos resultados concuerdan con las mejores prácticas de diseño de restaurantes en el mundo real, confirmando que la optimización realizada mediante técnicas de IA y simulación no solo encontró soluciones óptimas en términos de propinas, sino que también arrojó resultados que se alinean con la experiencia y la intuición.

Si bien este trabajo ha demostrado la eficacia de las técnicas utilizadas, el proyecto presenta algunas limitaciones que justifican futuras investigaciones. La optimización del restaurante se realizó en un entorno simulado, por lo que la aplicación de los resultados a un entorno real requiere una validación adicional. Además, el modelo actual no considera factores externos como la competencia o las fluctuaciones de la demanda, y se toman numerosas y extremas suposiciones para simplificar la simulación. El trabajo futuro debe abordar estos aspectos, buscando una mayor precisión y generalización del modelo, con el fin de obtener un sistema de optimización aún más completo y eficaz para la industria restaurantera. Asimismo, se pretende explorar el uso del Algoritmo A*, para búsqueda informada utilizando la heurística de distancia euclidiana, en cálculo de los caminos mínimos que toman los agentes, teniendo como premisa que por lo general los seres humanos se desplazan en la dirección más directa (en línea recta) hacia su objetivo.

Anexos

Ejemplo de una simulación de tan solo 5 minutos de tiempo de llegada de los clientes, en total

alrededor de 36 minutos de servicio.

Configuración:


```
    "single_simulation_grid": [
        [0, 0, 0, 0, 0, 3, 0, 0, 0, 0],
        [0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
        [0, 1, 2, 1, 1, 1, 1, 2, 1, 0],
        [0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
        [0, 1, 1, 1, 2, 1, 1, 1, 1, 0],
        [0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
        [0, 1, 1, 1, 1, 1, 1, 2, 1, 0],
        [0, 1, 2, 1, 1, 1, 1, 1, 1, 0],
        [0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
        [0, 0, 0, 0, 0, 4, 0, 0, 0, 0]
    ],
    "number_of_tables": 5,
    "number_of_waiters": 2,
    "simulation_duration": 300,
    "arrival_rate": 0.003,
```


Verbose:

```
Time: 35 Event: CustomerArrives
        Customer 0 started walking [(9, 5), (8, 5), (7, 5), (6, 5), (5, 4), (4, 3), (3, 3), (2, 2)] at time 35.
Time: 42 Event: CustomerSits
        Customer 0 stoped walking at time 42, position (2, 2).
Time: 230 Event: CustomerArrives
        Customer 1 started walking [(9, 5), (8, 5), (7, 5), (6, 5), (5, 5), (4, 5), (3, 6), (2, 7)] at time 230.
Time: 237 Event: CustomerSits
        Customer 1 stoped walking at time 237, position (2, 7).
Time: 239 Event: CustomerOrders
        Waiter 0 received new Request: take order of Customer 0, at time 239.
        Customer 0 started waiting at time 239.
        Waiter 0 started walking [(0, 5), (1, 4), (2, 3), (2, 2)] at time 239.
Time: 242 Event: WaiterStartsTakingOrder
        Waiter 0 stoped walking at time 242, position (2, 2).
        Customer 0 stoped waiting at time 242 (waited for 3s).
Time: 322 Event: CustomerOrders
        Waiter 1 received new Request: take order of Customer 1, at time 322.
        Customer 1 started waiting at time 322.
        Waiter 1 started walking [(0, 5), (1, 6), (2, 7)] at time 322.
Time: 324 Event: WaiterStartsTakingOrder
        Waiter 1 stoped walking at time 324, position (2, 7).
        Customer 1 stoped waiting at time 324 (waited for 2s).
Time: 335 Event: WaiterTakesOrder
        Waiter 0 finished Request: take order of Customer 0, at time 335.
        Waiter 0 received new Order of Dish 0 from Customer 0, at time 335.
        Customer 0 started waiting at time 335.
        Waiter 0 received new Request: return to the kitchen, at time 335.
        Waiter 0 started walking [(2, 2), (1, 3), (1, 4), (0, 5)] at time 335.
Time: 338 Event: WaiterReturnsToKitchen
        Waiter 0 stoped walking at time 338, position (0, 5).
        Waiter 0 finished Request: return to the kitchen, at time 338.
        The kitchen received a new Order of Dish 0 from Customer 0, at time 338.
        The kitchen started cooking Order of Dish 0 from Customer 0, at time 338.
Time: 488 Event: WaiterTakesOrder
        Waiter 1 finished Request: take order of Customer 1, at time 488.
        Waiter 1 received new Order of Dish 0 from Customer 1, at time 488.
        Customer 1 started waiting at time 488.
        Waiter 1 received new Request: return to the kitchen, at time 488.
        Waiter 1 started walking [(2, 7), (1, 6), (0, 5)] at time 488.
Time: 490 Event: WaiterReturnsToKitchen
        Waiter 1 stoped walking at time 490, position (0, 5).
        Waiter 1 finished Request: return to the kitchen, at time 490.
        The kitchen received a new Order of Dish 0 from Customer 1, at time 490.
        The kitchen started cooking Order of Dish 0 from Customer 1, at time 490.
Time: 981 Event: KitchenPreparesOrder
        The kitchen finished cooking Order of Dish 0 from Customer 0, at time 981.
        Waiter 0 took prepared Order of Dish 0 from Customer 0, at time 981.
        Waiter 0 received new Request: deliver Dish 0 to Customer 0, at time 981.
```

```
        Waiter 0 started walking [(0, 5), (1, 4), (2, 3), (2, 2)] at time 981.
Time: 984 Event: WaiterDeliversDish
        Waiter 0 stoped walking at time 984, position (2, 2).
        Customer 0 stoped waiting at time 984 (waited for 649s).
        Waiter 0 finished Request: deliver Dish 0 to Customer 0, at time 984.
        Waiter 0 received new Request: return to the kitchen, at time 984.
        Waiter 0 started walking [(2, 2), (1, 3), (1, 4), (0, 5)] at time 984.
Time: 987 Event: WaiterReturnsToKitchen
        Waiter 0 stoped walking at time 987, position (0, 5).
        Waiter 0 finished Request: return to the kitchen, at time 987.
Time: 1061 Event: KitchenPreparesOrder
        The kitchen finished cooking Order of Dish 0 from Customer 1, at time 1061.
        Waiter 0 took prepared Order of Dish 0 from Customer 1, at time 1061.
        Waiter 0 received new Request: deliver Dish 0 to Customer 1, at time 1061.
        Waiter 0 started walking [(0, 5), (1, 6), (2, 7)] at time 1061.
Time: 1063 Event: WaiterDeliversDish
        Waiter 0 stoped walking at time 1063, position (2, 7).
        Customer 1 stoped waiting at time 1063 (waited for 575s).
        Waiter 0 finished Request: deliver Dish 0 to Customer 1, at time 1063.
        Waiter 0 received new Request: return to the kitchen, at time 1063.
        Waiter 0 started walking [(2, 7), (1, 6), (0, 5)] at time 1063.
Time: 1065 Event: WaiterReturnsToKitchen
        Waiter 0 stoped walking at time 1065, position (0, 5).
        Waiter 0 finished Request: return to the kitchen, at time 1065.
Time: 1900 Event: CustomerFinishesEating
        Waiter 0 received new Request: collect the bill of Customer 0, at time 1900.
        Customer 0 started waiting at time 1900.
        Waiter 0 started walking [(0, 5), (1, 4), (2, 3), (2, 2)] at time 1900.
Time: 1903 Event: WaiterDeliversBill
        Waiter 0 stoped walking at time 1903, position (2, 2).
        Customer 0 stoped waiting at time 1903 (waited for 3s).
Time: 2004 Event: CustomerPays
        Customer 0 waited a total of 655s, left 18.657038090249195% of tip ($3.7314076180498 39), at time 2004.
        Customer 0 started walking [(2, 2), (3, 3), (4, 3), (5, 4), (6, 5), (7, 6), (8, 6), (9, 5)] at time 2004.
        Waiter 0 started walking [(2, 2), (1, 3), (1, 4), (0, 5)] at time 2004.
Time: 2007 Event: WaiterCleansTable
        Waiter 0 stoped walking at time 2007, position (0, 5).
        Waiter 0 finished Request: collect the bill of Customer 0, at time 2007.
Time: 2011 Event: CustomerLeaves
        Customer 0 stoped walking at time 2011, position (9, 5).
Time: 2116 Event: CustomerFinishesEating
        Waiter 0 received new Request: collect the bill of Customer 1, at time 2116.
        Customer 1 started waiting at time 2116.
        Waiter 0 started walking [(0, 5), (1, 6), (2, 7)] at time 2116.
Time: 2118 Event: WaiterDeliversBill
        Waiter 0 stoped walking at time 2118, position (2, 7).
        Customer 1 stoped waiting at time 2118 (waited for 2s).
Time: 2179 Event: CustomerPays
```

```
        Customer 1 waited a total of 579s, left 21.93974971558589% of tip ($4.387949943117178), at time 2179.
        Customer 1 started walking [(2, 7), (3, 8), (4, 8), (5, 8), (6, 8), (7, 7), (8, 6), (9, 5)] at time 2179.
        Waiter 0 started walking [(2, 7), (1, 6), (0, 5)] at time 2179.
Time: 2181 Event: WaiterCleansTable
        Waiter 0 stoped walking at time 2181, position (0, 5).
        Waiter 0 finished Request: collect the bill of Customer 1, at time 2181.
Time: 2186 Event: CustomerLeaves
        Customer 1 stoped walking at time 2186, position (9, 5).
Total tips: 8.119357561167018
```