Simulación Basada en Agentes de Tráfico Urbano con Programación Paralela

1. Introducción

La simulación basada en agentes (Agent-Based Modeling, ABM) es una metodología empleada para estudiar sistemas complejos en los cuales el comportamiento global surge a partir de interacciones locales entre entidades autónomas. En el caso del tráfico urbano, los vehículos interactúan entre sí y con infraestructura vial (calles e intersecciones), produciendo fenómenos emergentes como congestión, cuellos de botella y variaciones dinámicas del flujo.

El enunciado del proyecto plantea dos retos principales:

• Construir un **modelo secuencial** ABM de tráfico urbano con reglas locales explícitas.

• Implementar y evaluar una versión **paralela** que permita mejorar el rendimiento computacional en simulaciones de tamaño significativo, además de medir métricas del sistema y métricas computacionales (tiempo, speedup y eficiencia).

Este informe describe el modelo propuesto por el enunciado, documenta los supuestos y decisiones necesarios para implementarlo, presenta una implementación secuencial y una implementación paralela (paralelización por agentes), y reporta resultados experimentales con métricas del sistema y de desempeño computacional.

2. Descripción del Modelo

Esta sección se basa en los requisitos del enunciado (Secciones 4–6).

2.1 Representación espacial

El entorno se modela como una **rejilla bidimensional** de celdas. Cada celda representa uno de los siguientes tipos:

• **Calle transitable**: representada por el símbolo (.)

• **Intersección con semáforo**: representada por el símbolo (+)

• **Bloque urbano (no transitable)**: representada por el símbolo (#)

Las calles se representan como secuencias contiguas de celdas transitables, con múltiples celdas de longitud entre intersecciones consecutivas. El enunciado asume calles bidireccionales, por lo que cada celda transitable tiene capacidad para **dos vehículos**, uno en cada dirección.

Las intersecciones aparecen únicamente donde se cruzan calles perpendiculares y son los únicos puntos donde existen semáforos.

En esta implementación, además de respetar los símbolos del enunciado, se aplican **validaciones adicionales** al cargar la rejilla desde archivo (decisión técnica para evitar configuraciones inconsistentes). En particular:

• **Borde completamente bloqueado (#)**: se exige que todo el borde sea #.

• **Intersecciones estrictas**: cada + debe tener conectividad horizontal (E u O transitable) y vertical (N o S transitable).

• **Segmentos de calle rectos**: cada . Debe tener conectividad de grado 2 y pertenecer a un único eje (sin giros sobre .).

2.2 Agentes (vehículos)

Cada agente representa un vehículo e incluye como mínimo:

• **Posición** en la rejilla.

• **Dirección** de movimiento.

• **Estado de avance o detención**.

El número de agentes (**N**) es un parámetro. La ubicación inicial de los agentes se realiza en **lugares aleatorios** (según el enunciado).

2.3 Semáforos

Cada intersección posee un semáforo que alterna el paso de vehículos en direcciones perpendiculares. El enunciado permite que el cambio sea:

• **Periodico**

• **Adaptativo**

En esta implementación se usa control **periódico** (ver Sección 3 y 4), dejando el control adaptativo como posible extensión.

2.4 Reglas de interacción

En cada paso de tiempo (`tick`) de la simulación, el enunciado define el siguiente orden:
1. Los semáforos actualizan su estado.
2. Cada vehículo:
• Avanza a la siguiente celda si está libre y el semáforo lo permite.
• Se detiene si la celda siguiente está ocupada o el semáforo está en rojo.
• En intersecciones puede cambiar de dirección con cierta probabilidad.

El comportamiento global (p. ej., congestión) emerge de la aplicación repetida de estas reglas locales, junto con la capacidad bidireccional de las calles y la presencia de semáforos.

3. Supuestos y Decisiones del Modelo

El enunciado define la estructura general, pero deja abiertas varias ambigüedades. Esta sección documenta explícitamente lo que **no** queda completamente definido en el enunciado y fue necesario decidir para implementar el sistema.

3.1 Supuestos conceptuales

• **Actualización sincrónica por ticks**: se asume que todas las decisiones del tick se calculan con respecto al estado del tick anterior y se aplican de manera simultánea al finalizar el tick.
Esto facilita la coherencia y la comparación secuencial/paralela.
• **Interpretación operativa de "estado de detención"**: el enunciado exige que el vehículo tenga estado de avance/detención. Se implementa como un estado derivado por tick: un vehículo está "detenido" si no puede cambiar de celda en ese tick.
• **Definición operativa de "flujo promedio"**: el enunciado solicita "flujo promedio de vehículos", pero no fija una fórmula. Se define como el **promedio de vehículos que se movieron por tick** (vehículos que cambiaron de celda) a lo largo del horizonte temporal simulado.
• **Capacidad bidireccional por celda**: el enunciado indica "dos vehículos, uno en cada dirección". Se interpreta como:
  • Puede haber dos vehículos en la misma celda si sus direcciones son opuestas sobre el mismo eje (E/W o N/S).
  • No se permite mezclar simultáneamente un vehículo horizontal y uno vertical en la misma celda.
• **Resolución de conflictos cuando varios vehículos desean el mismo destino**: el enunciado no especifica cómo resolver colisiones de movimiento. Se adopta una regla determinista: gana el vehículo con menor identificador, y adicionalmente se aplica una exclusión por eje en cada celda destino para respetar la restricción de capacidad por eje.
• **Determinismo vs. estocasticidad**: el enunciado solicita inicialización aleatoria de agentes y giros probabilísticos en intersecciones, pero no exige determinismo. Se elige un diseño de aleatoriedad determinista para permitir reproducibilidad y comparación justa entre las versiones secuencial y paralela.
• **Interpretación de "el semáforo lo permite"**: el enunciado no especifica exactamente en qué transición se evalúa el semáforo. Se decide que el semáforo regula **la entrada a la intersección** (movimiento desde . hacía +). Esta interpretación se declara para eliminar ambigüedad operacional.

3.2 Decisiones técnicas

• **Lenguaje y librerías**: se implementa en Java (17) con una interfaz CLI.
• **Estructuras de datos**:
  • Estado de vehículos: arreglos para posición y dirección.
  • Ocupación: estructura indexada por celda y dirección.
  • Métricas: arreglos por tick (moved/stopped) y acumuladores.
• **Semáforos periódicos**: se implementa un semáforo con periodo fijo y alternancia entre verde horizontal y verde vertical.
• **Detalles de temporización del semáforo** (no prescritos por el enunciado):
  • Estado inicial: verde horizontal.
  • Alternancia: cuando `tick % period == 0` y `tick > 0`.
• **Motores de simulación separados**: `SequentialEngine` y `ParallelEngine` implementan el mismo modelo, cambiando únicamente la estrategia de ejecución.

• **Optimización del motor paralelo**: para cumplir el requisito del enunciado (“mejora de rendimiento en tamaño significativo”) fue necesario reducir overhead por tick. Se implementaron optimizaciones como:

- workers persistentes en vez de crear tareas por tick
- reducción de costos de reinicialización de estructuras temporales mediante técnicas de marcado por “stamp” para evitar recorridos completos de arreglos
- temporización comparable a la secuencial (tiempo de simulación dentro del loop de ticks).

3.3 Decisiones metodológicas

• **Control adaptativo no implementado**: se prioriza un modo periódico para la entrega base.
• **Selección de escenarios de benchmarking**: dado que el rendimiento paralelo depende del tamaño del problema, se diseñaron escenarios grandes (rejilla sintética) para evaluar “tamaño significativo” en términos de N y resolución espacial.
• **Énfasis en reproducibilidad**: se privilegia reproducibilidad (mismas decisiones aleatorias por vehículo/tick) sobre variaciones no controladas.

4. Implementación

Esta sección describe la implementación como realización del modelo descrito en el enunciado.

4.1 Lectura de rejilla

En cumplimiento de la Sección 6 del enunciado, la configuración espacial se lee desde un archivo de texto. Los símbolos válidos son . + #. Se valida que:
• el mapa sea rectangular
• contenga sólo símbolos permitidos
• los vehículos se muevan únicamente sobre . y +
• las intersecciones + representan cruces perpendiculares.
Adicionalmente (decisiones técnicas documentadas):
• se exige borde #
• se exige que los . sean segmentos rectos (sin giros ni cruces representados con .).

4.2 Semáforos

Se asigna un semáforo a cada intersección. Su estado alterna entre:
• **H_GREEN**: permite movimiento horizontal
• **V_GREEN**: permite movimiento vertical

4.3 Motor secuencial

El motor secuencial ejecuta, por tick:
1. Actualización de semáforos.
2. Cálculo de propuesta de movimiento por vehículo (avance o detención; posible giro en intersección).
3. Resolución de conflictos (ganadores).
4. Aplicación de movimientos de manera sincrónica.
Registra por tick la cantidad de vehículos que se movieron y que se detuvieron.

4.4 Motor paralelo (paralelización por agentes)

El enunciado permite la paralelización espacial o por agentes. Se implementa **paralelización por agentes**:
• Se divide el conjunto de vehículos en rangos y cada hilo procesa un rango.
• En cada tick:
  1. Los hilos computan propuestas en paralelo.
  2. Se resuelven conflictos de manera determinista.
  3. Los hilos aplican movimientos en paralelo.

La sincronización por tick se realiza mediante barreras para garantizar consistencia y evitar condiciones de carrera.

Limitación relevante para el desempeño: la fase de resolución de conflictos/ganadores reduce el paralelismo efectivo (por diseño y por costos de coordinación), lo cual limita el speedup máximo alcanzable.

4.5 Arquitectura del software, modelos de datos y algoritmos

El sistema se organiza en módulos para lectura de rejilla, semáforos, motores de simulación, reglas de movimiento, inicialización de vehículos, ocupación y recolección de métricas.

4.5.1. Componentes principales (módulos)

- **CLI / punto de entrada**: `Main`.
  - Permite ejecutar el motor en modo secuencial (`--mode seq`) o paralelo (`--mode par`).
  - Incluye modos experimentales: `--benchmark` (tabla resumen) y `--sweep` (barrido de parámetros).
- **Modelo espacial y lectura de rejilla**:
  - `GridLoader`: carga y valida el archivo de rejilla y construye el modelo interno.
  - `Grid`: representa la rejilla como un arreglo 1D (`idx = ywidth + x`) y provee consultas como `cellTypeAt`, `isTransitable` e índices de intersección.
- **Semáforos**:
  - TrafficLight + `TrafficLightState`: control periódico con alternancia H/V.
- **Simulación (motores)**:
  - `SimulationConfig`: agrupa los parámetros (`rejilla`, `N`, `ticks`, `seed`, `turnProb`, `periodo`, `modo`, `threads`, `salida`).
  - `SimulationEngine`: interfaz.
  - `SequentialEngine`: referencia para corrección del modelo.
  - `ParallelEngine`: misma lógica del modelo, ejecutada con paralelización por agentes.
- **Lógica local de movimiento**:
  - `MoveRules`: calcula la propuesta de movimiento por vehículo (incluye giro probabilístico determinista, validación de semáforo y ocupación).
- **Inicialización de agentes**:
  - `VehicleInitializer`: coloca vehículos en posiciones/direcciones aleatorias con seed, respetando la capacidad de la rejilla.
  - `VehicleState`: estado mínimo por vehículo (celda y dirección).
- **Ocupación / capacidad por celda**:
  - `Occupancy`: representa ocupación con un arreglo `occ` indexado por `(cellIdx, dirIdx)` y aplica la regla de capacidad por eje.
- **Métricas y salida de datos**:
  - MetricsCollector: registra `moved`/`stopped` por tick y calcula promedios.
  - CsvTicksWriter: escribe series por tick (tick,moved,stopped).
  - `BenchmarkRunner` y `SweepRunner`: ejecutan repeticiones y producen CSVs resumen con tiempos, speedup y eficiencia.

4.5.2. Modelo de datos: dirección, ocupación y capacidad

El estado dinámico se divide en:

- **Estado por vehículo**: arreglos `cellIdx[vehicleId]` y `dirIdx[vehicleId]`.
- **Ocupación por celda-dirección**: un arreglo `occ` con longitud `cellCount4`.
  - Clave: `key = cellIdx4 + dirIdx`.
  - Valor: `vehicleId` o `-1` si está libre.

La función `Occupancy.canOccupy(occ, cellIdx, dirIdx)` implementa la interpretación operativa de "dos vehículos por celda":

- Se permite ocupar un slot si ese mismo `(cellIdx, dirIdx)` está libre.
- Además, se impone **exclusión por eje**:
  - Si el vehículo es horizontal, la celda debe no tener ocupación vertical (`NORTH`/`SOUTH`).
  - Si el vehículo es vertical, la celda debe no tener ocupación horizontal (`EAST`/`WEST`).

Esto garantiza consistencia del modelo bajo la decisión de no mezclar ejes dentro de una misma celda.

4.5.3. Algoritmo por tick (común a secuencial y paralelo)

En ambos motores se mantiene el mismo orden lógico por tick:

1. **Actualización de semáforos** (control periódico).
2. **Cómputo de propuestas** por vehículo:
   - Si el vehículo está en +, puede girar con probabilidad `turnProb`.
   - Calcula la celda destino como la celda adyacente según la dirección.
   - Si la transición es . → +, se valida que el semáforo permita la dirección.
   - Se valida ocupación/capacidad del destino.
3. **Resolución de conflictos** cuando múltiples vehículos desean ocupar el mismo destino:
   - Se elige el ganador deterministicamente (prioridad por menor `vehicleId`) sujeto a la exclusión por eje.
4. **Aplicación sincrónica** (actualización global):
   - Se actualizan posición/dirección.
   - Se actualiza ocupación con doble buffer (ver siguiente punto).

Para garantizar la actualización sincrónica, se usa **doble buffer** de ocupación (`occ` / `occNext`) y se intercambian al final del tick.

4.5.4. Aleatoriedad determinista (reproducibilidad)

Para inicialización:

• `VehicleInitializer` genera candidatos de ocupación, los baraja con `SplittableRandom`(seed) y ubica los vehículos respetando capacidad.

Para giros en intersección:

• `MoveRules` usa `DeterministicRng.unitDouble(seed, vehicleId, tick, salt)` para derivar valores pseudoaleatorios reproducibles.

Esta decisión elimina diferencias entre ejecuciones secuencial/paralela causadas por acceso concurrente a RNGs compartidos.

4.5.5. Algoritmo paralelo (por agentes) y sincronización

El motor paralelo divide el conjunto de vehículos en rangos y ejecuta el tick en fases, sincronizadas con una barrera

• **Fase A (propuestas)**: cada hilo calcula propuestas para su rango leyendo el estado compartido del tick anterior.
• **Fase B (resolución)**: una etapa central resuelve ganadores de manera determinista.
• **Fase C (aplicación)**: cada hilo aplica movimientos para su rango y escribe en `occNext`.

Para reducir overhead, el motor usa hilos persistentes y estructuras temporales con técnica de marcado ("`stamp`") para evitar re-inicializaciones completas por tick.

5. Métricas y Metodología Experimental

5.1 Métricas del sistema

Con base en la Sección 8.1 del enunciado, se reportan:

• **Flujo promedio**: promedio de vehículos movidos por tick.
• **Vehículos detenidos**: promedio de vehículos que no se movieron por tick.

5.2 Métricas computacionales

Con base en la Sección 8.2 del enunciado, se reportan:

• **Tiempo de ejecución secuencial** (ms).
• **Tiempo de ejecución paralelo** (ms).
• **Speedup**: razón entre tiempo secuencial y paralelo.
• **Eficiencia**: speedup dividido entre número de hilos.

5.3 Metodología

Se ejecuta un benchmark con:

• warm-up (no registrado) -repeticiones (para promedios y desviación estándar)- lista de hilos `P` (2, 4, 8, 10, etc.).

Parámetros constantes (a menos que se indique lo contrario):

• semilla: 42- probabilidad de giro: 0.2- periodo del semáforo: 10.
Los resultados se guardan en CSV (`data/summary_.csv`) para trazabilidad y generación de gráficas.
Sobre la medición de tiempo: el tiempo reportado corresponde al **tiempo de simulación dentro del loop de ticks**. En particular, no incluye compilación/arranque del proceso y, en el motor paralelo, no incluye la creación del pool de hilos antes de iniciar la medición. Se considera adecuado para comparar el costo del cómputo del modelo por tick entre versiones.

6. Resultados y Análisis

6.1 Comportamiento del sistema

En las ejecuciones se observa que:
• El flujo promedio depende fuertemente de la densidad (N vs capacidad)- el número de detenidos aumenta con la congestión
• la alternancia de semáforos introduce oscilaciones y colas locales.
Para ilustrar la evolución temporal (moved/stopped por tick) se recomienda generar gráficas a partir de un `ticks_.csv` de una ejecución representativa.
En este informe se presentan principalmente **promedios agregados** (por tick) debido a que la evidencia incluida proviene de ejecuciones de benchmark con salida resumida. El análisis de series temporales por tick se propone como extensión inmediata.

6.2 Desempeño computacional

Un punto crítico del enunciado (Sección 9) es que **la versión paralela debe mostrar mejora de rendimiento para simulaciones de tamaño significativo**.
En mapas medianos y N relativamente pequeño (limitado por la capacidad), el overhead de sincronización y costos de memoria pueden dominar, produciendo speedup < 1. Este comportamiento se observó en el escenario huge.txt (N=3000, ticks=1000–2000).
Resultados reales (archivo `data/summary_opt2_huge.csv`, N=3000, ticks=2000, reps=3):

P (hilos)	Tiempo paralelo medio (ms)	Speedup	Eficiencia
2	147.00	0.876	0.438
4	156.40	0.824	0.206
8	211.80	0.608	0.076
10	219.00	0.588	0.059

Interpretación crítica:
• El enunciado no exige que el paralelo supere al secuencial en todo caso; exige mejora en “tamaño significativo”.
• Estos resultados son importantes para no ocultar limitaciones: con problemas pequeños, el costo de coordinación y sincronización puede superar el trabajo útil.
En contraste, al incrementar significativamente N y la resolución espacial, el trabajo por tick crece y el paralelismo amortiza el overhead, observándose **speedup > 1**.

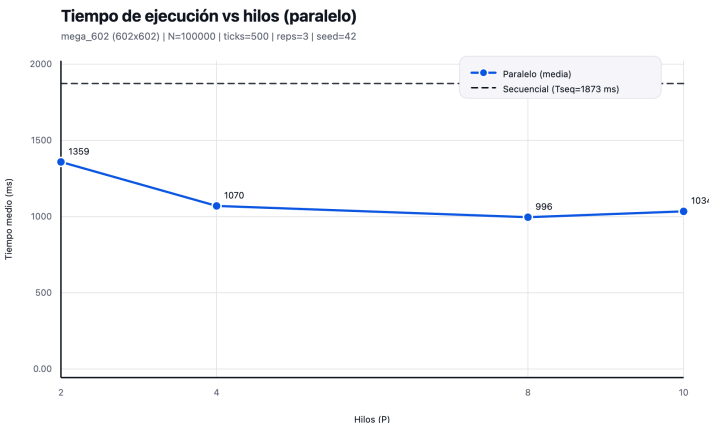
6.3 Evidencia de speedup

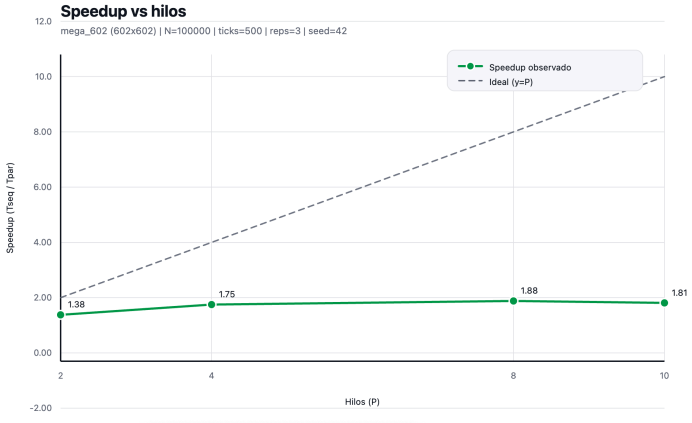
Se construyó una rejilla sintética grande `grids/mega_602.txt` (602x602) que cumple la lectura desde archivo externo y permite N alto. Se evaluó con:
N = 100000 vehículos, ticks = 500, P $\in \{2,4,8,10\}$, reps = 3
Resultados (archivo `data/summary_mega602_opt.csv`):

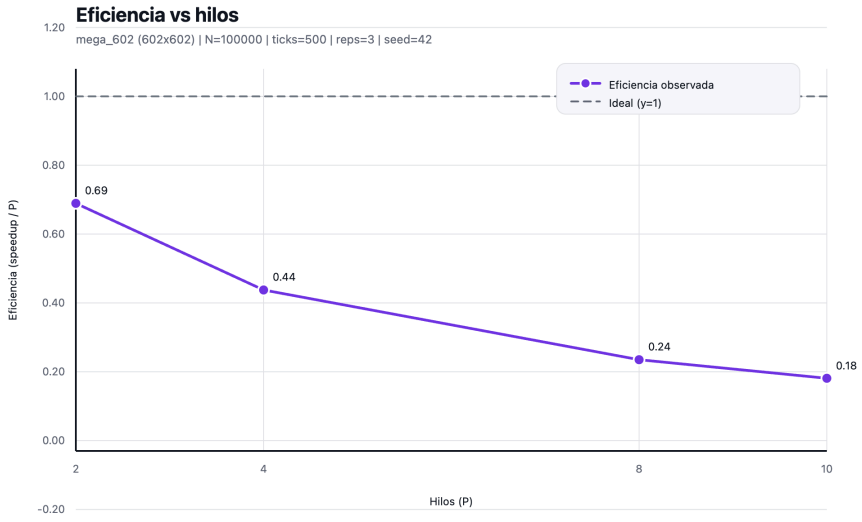
P (hilos)	Tiempo paralelo medio (ms)	Speedup	Eficiencia
2	1358.67	1.379	0.689
4	1070.00	1.750	0.438

8	996.00	1.881	0.235
10	1034.33	1.811	0.181

Interpretación:
• El speedup máximo observado en este punto experimental fue ~1.88 (P=8).
• La eficiencia decrece con P, consistente con límites por sincronización, fracción secuencial y ancho de banda de memoria.
• A pesar de ello, se cumple el requisito del enunciado: existe una configuración de tamaño significativo donde el paralelo supera al secuencial.







6.4 Limitaciones y Consideraciones del Análisis

• **Restricciones en la validación de rejillas**: La exigencia de bordes completamente bloqueados (#) y segmentos rectos sin giros sobre (.) mejora la detección de mapas inconsistentes, pero reduce la compatibilidad con configuraciones más flexibles permitidas por el enunciado. Esto limita los escenarios evaluados, aunque asegura consistencia en las pruebas.
• **Interpretación de semáforos**: Se evalúa el semáforo solo al ingresar a intersecciones (+) desde calles (.), sin regular explícitamente la salida o cruces internos. Esta aproximación simplifica la lógica, pero podría subestimar dinámicas complejas en intersecciones densas.
• **Manejo de capacidad y conflictos**: La exclusión por eje (no mezclar horizontal/vertical en una celda) y la resolución determinista por menor vehicleId garantizan consistencia y reproducibilidad, pero podrían sesgar el flujo en intersecciones comparado con un desempate aleatorio, afectando métricas como el flujo promedio.
• **Medición de tiempos**: Los tiempos se limitan al loop de simulación (excluyendo carga de archivos y setup de hilos), lo que

facilita comparaciones directas entre versiones, pero no refleja el tiempo total de ejecución en un pipeline completo.
• **Dependencia del hardware**: Los valores de speedup y eficiencia varían según el número de núcleos, políticas de scheduling y contención de memoria. Las repeticiones y desviaciones estándar reportadas mitigan esto, pero los resultados son específicos del entorno de prueba.

7. Conclusiones y Trabajo Futuro

7.1 Conclusiones

• Se implementó un modelo ABM de tráfico urbano en rejilla bidimensional con semáforos en intersecciones, cumpliendo los componentes requeridos por el enunciado (rejilla, agentes, reglas locales, lectura externa).
• Se implementó una versión paralela mediante **paralelización por agentes** y sincronización por tick, evitando condiciones de carrera mediante barreras y reglas deterministas de resolución de conflictos.
• Se reportaron métricas del sistema (flujo promedio, vehículos detenidos) y métricas computacionales (tiempos, speedup, eficiencia), y se generó evidencia experimental de **mejora de rendimiento** en un escenario de tamaño significativo.
• Se observaron también configuraciones donde el paralelismo **no mejora** (speedup < 1) cuando el tamaño del problema es reducido. Esto ilustra el trade-off entre overhead de paralelización y trabajo útil.

7.2 Limitaciones

• El enunciado permite semáforos adaptativos, pero la implementación usa control periódico.
• En escenarios medianos con N limitado por capacidad, el overhead puede impedir speedup > 1. Esto no contradice el enunciado siempre que exista evidencia en tamaño significativo; sin embargo, es relevante como limitación práctica.
• La validación de rejilla y la interpretación exacta de capacidad por celda son decisiones que deben declararse explícitamente (Sección 3).

7.3 Trabajo futuro

• Implementar semáforos adaptativos basados en la congestión local.
• Explorar paralelización adicional de la resolución de conflictos, o estrategias alternativas (p. ej., particionado espacial) para mejorar rendimiento en mapas medianos.
• Ampliar el conjunto de experimentos (barridos de N/ticks/P) y consolidar gráficas comparativas para caracterización más completa.
• Incorporar resultados por tick (series temporales) para caracterizar mejor la dinámica de congestión, complementando los promedios.