Tesis de Licenciatura
en Ciencias de la Computación

Simulación Distribuida Basada en Agentes Utilizando Balanceo Dinámico de la Carga

Tesista
Matías Tencer <mtencer@dc.uba.ar>
Director
Lic. Marcelo Gilman <mgilman@urbix.com.ar>
Co-Director
Lic. Esteban Mocskos <emocskos@dc.uba.ar>

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Diciembre 2006

Resumen

Aplicar la simulación computacional al problema de tráfico peatonal es un tema reciente y de gran interés. Permite la correcta elaboración de planes de evacuación, optimización del espacio y análisis de la calidad de vida en espacios altamente transitados, entre otras aplicaciones. Un problema de este tipo puede considerarse como un sistema autoorganizado, donde el comportamiento global emerge de las interacciones de las partes o individuos que lo forman. Plantear el problema de tráfico peatonal como un sistema autoorganizado, facilita un abordaje microscópico que permite una descripción detallada de cada peatón y sus interacciones, permitiendo de esta forma abordar una gran cantidad de situaciones. Este enfoque requiere la utilización de una gran cantidad de recursos computacionales, sin embargo, tiene la ventaja de ser altamente paralelizable. En este trabajo presentamos un modelo de simulación distribuida basada en agentes con balanceo dinámico de la carga utilizando técnicas de partición de grafos. Este modelo resulta ser eficiente para resolver el problema de tráfico peatonal en un cluster de computadoras. La utilización de balanceo dinámico de la carga permitió adaptarse a la dinámica de la simulación disminuyendo la sobrecarga del sistema debido a los tiempos ociosos de los nodos con menos carga. En este trabajo se presentan ejemplos donde con gran cantidad de peatones se obtuvieron aceleraciones del orden de 3.4 para un cluster de cuatro procesadores.

Abstract

Applying computer simulation to pedestrian traffic problems is a recent issue of great interest. Among other applications, it allows the correct evacuation plan design, space optimization and life quality analysis of great pedestrian transit places. This kind of problem could be considered an auto-organized system where the global behavior emerges from the interaction of its parts or its individuals. To consider the pedestrian traffic problem as an auto-organized system facilitates a microscopic treatment that allows a detailed description of each pedestrian and its interactions in order to afford a great quantity of situations. This technical method demands great computer resources. However, it presents the advantage of its high parallelization. In this thesis, we present an agent-based distributed simulation model with dynamic load balance, using the graph partition method. This is an efficient model to resolve the pedestrian traffic problem on a computer cluster. Using dynamic load balance allowed the adaptation to the dynamics of the simulation reducing system overcharge due to idle times in the nodes with less charge. In this study we show examples where we got a range of accelerations of 3.4 on a 4 processors cluster.

Agradecimientos

Quiero agradecer a Urbix por darme la posibilidad y el lugar para poder realizar esta tesis. A mi familia por estar siempre y apoyarme. A Andre por ser una persona muy especial, ayudarme y sobre todo escucharme en todo momento.

Índice general

Capítulo 1

Simulación Basada en Agentes

1.1. Introducción

La simulación computacional permite analizar grandes sistemas reales sin la necesidad de experimentar con ellos. Con la aparición de las computadoras digitales, esta metodología se ha ido introduciendo en la investigación científica en forma alternativa o complementaria a los métodos convencionales como la experimentación y el análisis teórico. Mediante la simulación computacional, se pueden analizar problemas donde la complejidad de los mismos imposibilitan un análisis teórico y analítico. Modelando y simulando se pueden resolver problemas donde no se cuenta con el sistema real, o que el experimentar sobre el mismo implica asumir situaciones peligrosas para los seres humanos. Con la simulación computacional se reducen costos, se aceleran los tiempos de obtención de resultados, se da la posibilidad de probar con distintos entornos y configuraciones de manera ágil y mucho más.

A pesar del continuo avance, en los sistemas monoprocesamiento existen limitaciones físicas (velocidad de la luz, leyes termodinámica y altos costos) que imposibilitan el crecimiento de los mismos [14]. La computación distribuida tiene un gran potencial para resolver problemas complejos que demanden una alta cantidad de recursos de procesamiento y de memoria. Tradicionalmente se utilizaban supercomputadoras, pero el aprovechamiento de los clusters de PCs ha resultado una buena alternativa por sus prestaciones, costos y flexibilidades.

La simulación de tráfico peatonal es un área de investigación reciente que deriva de la dinámica molecular [12]. Es de gran importancia para la correcta elaboración de planes de evacuación, optimización del espacio y análisis de la calidad de vida en

espacios altamente transitados, entre otras aplicaciones. Generalmente es deseable probar distintos entornos y realizar pruebas antes de contar con el escenario real. Para el caso de las evacuaciones o de sistemas de grandes multitudes realizar el análisis en el sistema real, por ejemplo a través de simulacros, tiene un alto costo. Inclusive en algunos casos es imposible de realizar debido a la necesidad plantear distintos escenarios, ataques de pánico, estampidas, etc. Existen distintos enfoques para resolver este problema. El enfoque macroscópico mediante el análisis de un flujo continuo, representa la aproximación más sencilla pero muy cruda ya que no permite reproducir una gran cantidad de fenómenos propios de los flujos granulares en general y de peatones en particular. En contraposición, el enfoque microscópico, donde se representa cada peatón en forma individual permite simular una gran variedad de situaciones imposibles de ser abordadas en forma macroscópica. De esta forma, modelando el comportamiento de cada individuo que compone el sistema se obtiene un comportamiento emergente de la multitud. La desventaja de este enfoque es que requiere una mayor cantidad de cómputos a realizar.

En los últimos años el paradigma de agentes se ha difundido siendo aplicado en diversos problemas [26], sin embargo todavía se encuentra en desarrollo y no ha llegado a un estado de madurez. El paradigma evolucionó a partir de disciplinas como la inteligencia artificial, la róbotica y la programación orientada a objetos. Un agente puede pensarse como una entidad capaz de percibir su entorno a través de sensores y transformarlo a través de actuadores.

Este paradigma tiene un gran potencial para resolver problemas como el de tráfico peatonal mediante un enfoque microscópico. Para resolver problemas de peatones de gran envergadura (miles de personas), donde no alcanza con una sola máquina, en este trabajo hemos introducido un modelo que permite resolver este problema en un cluster de computadoras. Mediante la aplicación de técnicas de partición de grafos al balanceo dinámico de carga en una plataforma de agentes, hemos logrado resolver este problema eficientemente.

En el resto de este capítulo introducimos los conceptos básicos de la simulación basada en agentes. En el capítulo 2 describimos la problemática de la simulación del tráfico peatonal, su marco teórico y un modelo conceptual para resolverlo. En el capítulo 3 presentamos el modelo computacional centralizado y el distribuido, propuestos en este trabajo, para abordar la problemática introducida anteriormente y en el capítulo 4 proponemos una estrategia de partición del espacio para lograr la distribución de agentes en un cluster de computadoras y lograr un balanceo dinámico de la carga. Fi-

nalmente en el capítulo 5 se presentan los resultados obtenidos y en el 6 las conclusiones y trabajo futuro. Adicionalmente se incluyen dos apéndices. El primero correspondiente a la teoría de partición de grafos que es utilizada como soporte para la partición del espacio, y el segundo a un glosario con las definiciones utilizadas en esta tesis.

1.2. Sistemas Autoorganizados

La idea que la dinámica de un sistema puede por sí misma incrementar el orden inherente del propio sistema tiene una larga historia. Antiguos atomistas y naturalistas creían que no era necesaria una "inteligencia diseñadora" para lograr un orden en la naturaleza y consideraban que las reglas ordinarias de la naturaleza eran suficientes para lograr este orden. Uno de los primeros en desarrollar esta idea fue Descartes en su libro -nunca publicado- Le Monde.

Estas ideas fueron abandonadas hasta comienzos del siglo 20, cuando distintos científicos, especialmente de la física y la química revivieron la idea. El término "autoorganización" parece haber sido introducido en 1947 por el psiquiatra e ingeniero W. Ross Ashby. El término fue utilizado por científicos del área de teoría general de sistemas, pero su uso se unificó cuando se adoptó el concepto en el área de sistemas complejos durante los '70 y '80.

¿Qué características comparten los sistemas autoorganizados? Según Steven Johnson en [20], "estos sistemas resuelven problemas recurriendo a masas de elementos relativamente no inteligentes en lugar de hacerlo recurriendo a un solo brazo ejecutor inteligente. Son sistemas ascendentes, no descendentes. Extraen su inteligencia de la base. [...] En estos sistemas, los agentes que residen en una escala comienzan a producir comportamientos que yacen en una escala superior a la suya: las hormigas crean colonias, los habitantes de una ciudad crean barrios, un software de reconocimiento de patrones aprende a recomendar libros."

Para F. Heylighen [21] la autoorganización es un proceso evolutivo en el cual la organización de un sistema se produce por fuerzas propias y con un pequeño impacto del entorno. La autoorganización se dispara por procesos internos llamados "fluctuaciones" o "ruido". Estos procesos generan una estructura que perdura en el tiempo. Este fenómeno fue denominado "orden a partir de ruido" por Heinz von Foerster y como "orden a partir de fluctuaciones" por Ilya Prigogine.

1.3. Propiedades emergentes

Consideremos dos puntos en un plano. Estos puntos tendrán una distancia entre ellos. Esta distancia no es en sí misma una propiedad de cada punto sino de su relación.

En forma general, los sistemas pueden poseer propiedades cuyas componentes individuales no poseen. Una propiedad sobre un conjunto de agentes se dice emergente cuando la misma existe sobre el conjunto y no sobre los agentes individuales. Según F. Heylighen en [22], una propiedad emergente es irreducible, pertenece al *todo* y no puede ser definida en términos de las *partes*.

Las propiedades emergentes pueden no sólo definirse sobre agentes sino también sobre otras propiedades emergentes. Continuando con el ejemplo de los puntos en el plano, tres puntos serán equidistantes si sus distancias medidas de a pares son iguales. La propiedad *equidistancia* se define sobre la propiedad *distancia* que se define sobre los puntos en el plano.

En los sistemas complejos, como colonias de insectos o animales, urbes u organizaciones humanas, las propiedades emergentes surgen cuando el sistema alcanza un cierto umbral de diversidad, organización y conectividad. En estos contextos, las propiedades emergentes son difícilmente predecibles o deducidas de las propiedades de los individuos.

Las estructuras emergentes son sistemas generados a partir de las interacciones de distintos agentes que al interactuar cada uno con un entorno determinado mediante ciertas reglas, generan un "orden" que se reconoce como una nueva estructura o agente.

El comportamiento de los mercados de valores es un típico ejemplo de estructura emergente. Como sistema, regula los precios relativos de empresas a lo largo del mundo. Sin embargo no existe un líder o "marcapasos" que controle la totalidad del mercado. Cada agente o inversor posee conocimiento de un número limitado de compañías dentro de su portfolio y debe seguir las regulaciones del mercado. A partir de las interacciones de los inversores individuales emerge el mercado como entidad en sí misma.

1.4. Comportamientos

Los agentes como personas, insectos, animales o robots poseen comportamientos. Una definición ampliamente aceptada de comportamiento es aquella que establece que es un mapeo de estímulos en acciones para distintos entornos. Bajo este enfoque, podemos considerar a un comportamiento como una propiedad emergente de los agentes,

en tanto que sólo se manifiesta en la interacción de los agentes con su entorno y no sobre cada uno de los mismos de manera aislada.

Los comportamientos suelen clasificarse en deliberativos o reactivos. Según R. C. Arkin [23], un comportamiento deliberativo "es más adecuado para integrar el conocimiento del mundo y para que un agente elabore un plan"; por otro lado, un comportamiento reactivo "es adecuado para tratar con la inmediatez de los datos sensoriales".

En los comportamientos deliberativos las acciones son la consecuencia de un proceso de "deliberación" o razonamiento. Son generalmente asociados a niveles cognitivos superiores y su rango de acción es de mediano a largo alcance, en términos de entorno del agente. En cambio, en los reactivos las acciones son disparadas inmediatamente como consecuencia de los estímulos y son generalmente asociados a comportamientos con un rango de acción relativamente cercano al agente o local, como por ejemplo el control motor y la percepción.

1.5. Agentes como Paradigma Computacional

El término *agente* es utilizado en muchos contextos distintos. Si bien no existe aún consenso sobre el término en contextos computacionales, una definición típica establece que "un agente es un sistema informático situado en un entorno y que es capaz de realizar acciones de forma autónoma para conseguir sus objetivos de diseño" [24].

Si bien esta definición contempla algunas de las características de un agente, muchos sistemas informáticos convencionales satisfacen esta definición.

En este trabajo ya hemos utilizado el término *agente* como un elemento de un sistema autoorganizado. Es precisamente éste el enfoque que adoptaremos. Definiremos entonces a un agente como el elemento computacional que representa un elemento de un sistema autoorganizado [25]. Los *agentes computacionales* se corresponden uno a uno con los *agentes reales*, como hormigas o personas. Podemos pensar entonces que *agentes* es un paradigma computacional que permite describir fácilmente y resolver eficientemente problemas en el dominio de los sistemas autoorganizados.

Los agentes poseen comportamientos que permiten su interacción con el entorno que los rodea. Sean estos dados a priori o aprendidos mediante algún mecanismo de adaptación, los comportamientos constituyen el núcleo sobre el que se trabajará en el paradigma de agentes.

Además de clasificar a los agentes como reactivos o deliberativos según su com-

portamiento, los agentes se dicen proactivos si pueden tomar alguna acción sin recibir estímulo alguno del entorno. Si bien esto parece una trasgresión a la definición de comportamiento, la carencia aparente de un estímulo para la elección de una acción puede ser salvada si consideramos que un agente puede autoestimularse. En este caso la proactividad es un caso particular de reactividad.

1.6. Simulación Basada en Agentes

La simulación basada en agentes es una eficaz herramienta para resolver problemas asociados a sistemas autoorganizados. Una de sus principales ventajas es que el lenguaje computacional se corresponde con el modelo conceptual: agente, entorno, estímulo, acción, sensores.

Con el fin de implementar la proactividad, muchos de los frameworks de agentes asignan a cada agente un thread (hilo de ejecución en un procesador) o proceso (ver por ejemplo Jade[1]). Una consecuencia es que en los procesadores actuales, un sistema basado en estos mecanismos no soporta más que unas decenas o, a lo sumo, unas pocas centenas de agentes simultáneamente.

Este esquema no se ajusta bien para simular sistemas con miles de elementos. Otro abordaje consiste en combinar los agentes con un sistema de simulación por eventos discretos (ver por ejemplo Swarm[2]). En estos frameworks los agentes sólo pueden reaccionar a los estímulos de su entorno y su proactividad deberá ser "simulada", como se menciona en la sección anterior. Al acotarse la cantidad de procesos utilizados, bajo este esquema es posible simular sistemas con miles de agentes.

[1]Jade es una de las librerías de agentes más conocidas. http://jade.tilab.com/

[2]Swarm es una librería de agentes que utiliza simulación por eventos. http://xenia.media.mit.edu/ñelson/research/swarm

Capítulo 2

Simulación Peatonal

La simulación del tráfico peatonal permite modelar el comportamiento de una multitud de personas moviéndose en un espacio determinado. Un modelo de tráfico peatonal se puede dividir en dos aspectos [13]:

- simulación del sistema físico: cálculo de interacciones entre las personas y movimiento de las mismas,
- estrategia de planificación: elección del destino y la ruta de navegación.

El tráfico peatonal puede ser pensado como un sistema autoorganizado donde las partes que lo componen son los peatones. Es un problema altamente dinámico donde los peatones cambian su estado continuamente, requiriendo una gran cantidad de cálculos. Cada peatón tiene un comportamiento propio y a partir de las interacciones con su entorno (otros peatones, el lugar por donde camina, etc.) emerge un comportamiento de la multitud. Por ejemplo, en [28, 29] se muestra que existe una relación entre la densidad de peatones y la velocidad de los mismos. Podemos considerar que un peatón posee rasgos comportamentales reactivos (control motor, percepción) y deliberativos (planificación).

En este trabajo nos concentraremos en los aspectos de la simulación del sistema físico. Este problema puede ser pensado como un caso particular de la dinámica molecular. Las personas son representadas por partículas autoimpulsadas interactuando a partir de un modelo de fuerzas. En la siguiente sección introduciremos la simulación de dinámica molecular y a continuación el modelo físico que utilizamos en este trabajo.

2.1. Simulación de Dinámica Molecular

> *Dado por un instante una inteligencia tal que pudiera comprender todas las fuerzas que animan la naturaleza y las respectivas situaciones de cada unos de los seres que la componen - una inteligencia lo suficientemente vasta para considerar el análisis de esta data - podrá abarcar en la misma fórmula desde los movimientos de los cuerpos más grandes hasta los movimientos de los átomos más livianos; para ella nada será incierto y tanto el futuro como el pasado sería presente ante sus ojos.*
>
> Laplace, 1814

La simulación de *dinámica molecular* (MD) consiste en calcular el movimiento individual de cada partícula que modela un sólido, un líquido o un gas. El movimiento se describe calculando los cambios de posiciones, velocidades y orientaciones a lo largo del tiempo [1, 2].

Los métodos de MD aparecieron por primera vez en los '50 pero comenzaron a recibir atención recién a mediados de los '70, cuando las computadoras digitales se convirtieron en suficientemente poderosas como para resolver todos los cálculos involucrados [1]. La esencia de MD es resolver el problema de N cuerpos de la mecánica clásica, la cual es una evolución de una vieja idea en la ciencia. Básicamente la idea es que el comportamiento de un sistema se puede computar si tenemos las partes del mismo, el conjunto inicial de condiciones y las fuerzas de interacciones. Desde el tiempo de Newton hasta el presente, esta interpretación determinística de la mecánica de la naturaleza ha dominado la ciencia [1]. El tema de simulación de N cuerpos siempre ha sido importante, y las razones de su importancia fueron evolucionando. Hoy en día, lo que se intenta relacionar es la dinámica colectiva con la dinámica de una única partícula, con el objetivo de poder explicar el complicado comportamiento de una gran colección de partículas a partir de su movimientos individuales.

La simulación a escala molecular comprende tres pasos: a) modelar las partículas individuales, b) simular el movimiento del conjunto de partículas, c) analizar los datos obtenidos de la simulación. La primer etapa comprende determinar qué tipo de propiedades nos interesan estudiar, dentro de qué rango de parámetros y con qué precisión. En función de ello debemos determinar con qué tipo de partículas contamos y cuáles son las fuerzas de interacción entre las mismas. La información que genera una corrida de MD es la posición y velocidad de cada partícula del sistema en cada instante de

tiempo. Empleando técnicas tradicionales de la mecánica estadística podemos pasar de esta información microscópica a la obtención de magnitudes macroscópicas similares a las observadas experimentalmente.

La ejecución de una simulación comprende, a partir de una configuración inicial, calcular las fuerzas de interacciones entre las partículas e integrar las ecuaciones de movimiento de Newton para cada una, dada por la siguiente fórmula:

$$F_i(t) = m_i \ddot{r}_i(t) \tag{2.1}$$

donde F_i es la fuerza resultante sobre la partícula i causada por las otras $N-1$ partículas, m_i la masa, $\ddot{r}$ es la segunda derivada de la trayectoria, que representa la aceleración.

Esto da lugar a N ecuaciones diferenciales acopladas cuya solución es posible de encontrar únicamente discretizando. Para resolver esto existen diversos algoritmos, uno de ellos es el llamado *algoritmo de Verlet*, que es uno de los más simples y efectivos. Para deducirlo, partimos del desarrollo de la serie de Taylor de la función de trayectoria de una partícula,

$$r(t+\Delta t) = r(t) + v(t)\Delta t + \frac{F(t)}{2m}\Delta t^2 + \frac{\Delta t^3}{3!}r'''(t) + O(\Delta t^4) \tag{2.2}$$

donde $v(t)$ es la función de velocidad, y $F(t)$ es la fuerza que se le aplica a la partícula en el instante t. Del mismo modo,

$$r(t-\Delta t) = r(t) - v(t)\Delta t + \frac{F(t)}{2m}\Delta t^2 - \frac{\Delta t^3}{3!}r'''(t) + O(\Delta t^4) \tag{2.3}$$

Sumando ambos desarrollos, obtenemos

$$r(t+\Delta t) + r(t-\Delta t) = 2r(t) + \frac{F(t)}{m}\Delta t^2 + O(\Delta t^4) \tag{2.4}$$

Así, la nueva posición r buscada, en el tiempo $t+\Delta t$, viene dada por

$$r(t+\Delta t) \approx 2r(t) - r(t-\Delta t) + \frac{F(t)}{m}\Delta t^2 \tag{2.5}$$

El error estimado que contiene la nueva posición es del orden Δt^4, donde Δt es el paso de discretización del tiempo. Nótese que para calcular la nueva posición $r(t+\Delta t)$ sólo es necesario conocer la posiciones $r(t-\Delta t)$ y $r(t)$, y la fuerza $F(t)$; mientras que la velocidad no es necesaria. Sin embargo, ésta la podemos calcular a partir de

$$r(t + \Delta t) - r(t - \Delta t) = 2v(t)\Delta t + O(\Delta t^3) \tag{2.6}$$

de donde se deduce

$$v(t) = \frac{r(t + \Delta t) - r(t - \Delta t)}{2\Delta t} + O(\Delta t^2) \tag{2.7}$$

2.2. Modelo Conceptual

Adoptamos el modelo conceptual introducido por Dirk Helbing et al. en [12] para situaciones de pánico. El modelo esta basado en una mezcla de factores sociopsicológicos y fuerzas físicas que influencian el comportamiento de las personas en una multitud. En este modelo se simplificó el problema de tráfico peatonal al comportamiento de "ir a un lugar", donde los cálculos involucrados son las fuerzas de interacción y la integración de la ecuación de Newton. De esta forma, el modelo es un caso particular de simulación de dinámica molecular. La trayectoria r_i de cada peatón p_i es calculada a partir de la suma de tres fuerzas con distintas propiedades: "Fuerza del Deseo" (FD), "Fuerza Social" (FS) y "Fuerza Granular" (FG). La primera modela el deseo de una persona de llegar a un cierto objetivo a una determinada velocidad. La segunda modela las interacciones sociales con los vecinos, siendo una fuerza de repulsión que hace que los peatones se mantengan alejados de los que lo rodean. La última modela las fuerzas físicas que existen cuando dos partículas están en contacto.

Las expresión de la fuerza del deseo esta dada por

$$FD(i) = m_i \frac{v_{di} e_i - v_i}{\tau} \tag{2.8}$$

donde m_i es la masa del peatón, v_i y v_{di} son la velocidad real y deseada respectivamente, e_i es el versor que apunta al objetivo y τ es una constante relacionada con el tiempo que tarda el peatón en alcanzar su velocidad deseada. Cuanto más chica sea esta constante, más alto es el valor de la fuerza del deseo, por lo que se alcanzará la velocidad v_{di} más rápidamente.

La fuerza social se calcula a partir de una sumatoria de fuerzas dada por la siguiente expresión:

$$fs(i, j) = A \exp\left(\frac{-\epsilon_{ij}}{B}\right) e_{ij}^n \tag{2.9}$$

donde $\epsilon_{ij} = r_{ij} - (R_i + R_j)$ con r_{ij} distancia entre los centros de masa de p_i y p_j y R el radio del peatón. A y B son constantes que determinan la intensidad y rango de la interacción social, y e^n_{ij} es el versor que apunta desde peatón p_j al p_i (es la dirección "Normal" entre dos peatones).

Resultando una fuerza social total sobre el peatón p_i dada por la siguiente expresión:

$$FS_P(i) = \sum_{j \in P, j \neq i} fs(i,j) \tag{2.10}$$

donde P es el conjunto de todos los peatones.

Por último la fuerza granular, que también se calcula a partir de una sumatoria de fuerzas, se refleja en esta otra expresión:

$$fg(i,j) = [-\epsilon_{ij} k_n e^n_{ij} + (v^t_{ij} \epsilon^t_{ij} k_t) e^t_{ij}] g(\epsilon_{ij}) \tag{2.11}$$

resultando una fuerza granular total sobre el peatón p_i dada por la siguiente expresión:

$$FG_P(i) = \sum_{j \in P, j \neq i} fg(i,j) \tag{2.12}$$

donde el versor tangencial (e^t_{ij}) indica la correspondiente dirección perpendicular, k_n y k_t son las constantes de restauración elástica normal y tangencial respectivamente, v^n_{ij} y v^t_{ij} son las proyecciones normal y tangenciales de la velocidad relativa vista desde p_j ($v_{ij} = v_i - v_j$), y la función $g(\epsilon_{ij})$ es $g = 1$ si $\epsilon_{ij} < 0$ o $g = 0$ en caso contrario.

Las interacciones de los peatones con los obstáculos son computadas de la misma forma, a través de la fuerza social y granular. Para realizar los cálculos, para cada obstáculo se toma el punto más cercano a cada peatón y se calculan las fuerzas que influyen sobre cada peatón. Se puede definir $FS_O(i)$ como la sumatoria de las fuerzas sociales entre el peatón i y cada obstáculo del conjunto de obstáculos O y, en forma análoga, $FG_O(i)$ para la fuerza granular.

Concluimos expresando la fuerza total que influye sobre un peatón i como

$$F(i) = FD(i) + FS_P(i) + FG_P(i) + FS_O(i) + FG_O(i) \tag{2.13}$$

Los parámetros utilizados en la simulaciones para obtener un comportamiento que se asemeje a la realidad son los siguientes:

Parámetro	Valor
τ	0,5 s
A	2000 N
B	0,08 m
k_t	$2 * 10^5$ N/m
k_n	10^5 kg/m/s
m_i	$\approx$ 74 kg $\forall i$
R_i	$\approx$ 0,255 m $\forall i$
v_{di}	1-2 m/s $\forall i$

Capítulo 3

Modelo Computacional

Como vimos anteriormente el problema de simulación del tráfico peatonal puede ser pensado como un sistema autoorganizado, para los cuales el paradigma de agentes es adecuado. De esta forma nos permite modelar el problema a partir de las partes que lo forman: los peatones. Un peatón del mundo real es modelado por un agente en el modelo computacional. Utilizaremos el término peatón y agente en forma indistinta. Aunque en este trabajo un peatón no tiene un comportamiento deliberativo, y puede ser pensado únicamente como un punto en el espacio con determinado radio y masa, en ningún momento queremos perderlo como unidad conceptual y ser la partícula más pequeña que compone nuestro sistema. En un futuro el simulador podría ser ampliado y contemplar un comportamiento mucho mas complejo y requerir una mayor cantidad de cálculos. Utilizar el paradigma de agentes permite agregar este comportamiento de una forma sencilla.

En este capítulo analizamos una técnica para poder lograr una distribución del conjunto de las unidades que forman el sistema en un cluster de computadoras. En la primer sección se plantea un modelo centralizado donde se introduce cuál es el algoritmo utilizado para resolver el problema en sí, mientras que en la siguiente se presenta el modelo computacional distribuido.

3.1. Modelo Computacional Centralizado

La primera y más simple aproximación para resolver el problema planteado en el capítulo anterior, se ve en el algoritmo 1. Para cada instante de tiempo desde 1 hasta un tiempo final (t_f) a partir de un estado inicial, se calculan las fuerzas de interacción

entre todos los peatones y los obstáculos basadas en las posiciones del instante anterior y, luego, se actualizan las posiciones de los mismos. Sin embargo, esta no es la mejor solución ya que tiene un orden de complejidad cuadrático con respecto a la cantidad de peatones (considerando que la cantidad de obstáculos es significativamente menor en todos los casos).

Algoritmo 1

1: **Para** $t = 1, ..., t_f$ cada Δt **Hacer**
2: **Para Todo** peatón p **Hacer**
3: Calcular $F(p)$
4: **Fin Para**
5: **Para Todo** peatón p **Hacer**
6: Actualizar la posición de p
7: **Fin Para**
8: **Fin Para**

El paso 3 del algoritmo implica calcular la fuerza correspondiente según la fórmula 2.13, mientras que el paso 6 comprende evaluar la expresión 2.5 para el tiempo correspondiente. Es necesario calcular la fuerzas que influyen sobre todos los peatones antes de actualizar las posiciones de los mismos. Todas las fuerzas deben ser calculadas sobre las posiciones correspondientes al mismo instante de tiempo y no es posible actualizar dicha posición hasta saber que no es necesaria para calcular otra interacción.

Como las fuerzas de interacción decrecen a medida que la distancia entre peatón-peatón o peatón-obstáculo aumenta, a partir de cierta distancia las mismas son despreciables. Por este motivo, sólo es necesario calcular las fuerzas en las cercanías. Llamamos *vecinos* $V_d^t(p)$ al conjunto de peatones y obstáculos que se encuentran a una distancia menor o igual que d con respecto a la posición del peatón p para el instante de tiempo t, y *radio de influencia* ri a la distancia a partir de la cual las fuerzas son despreciables. Luego, para cada peatón sólo se calculan las fuerzas correspondientes a las interacciones con los elementos del conjunto de vecinos.

La fuerza $F(p)$ que actúa sobre el peatón p calculada en la expresión 2.13 puede ahora ser aproximada por la fuerza $F_{V_{ri}^t}(p)$, calculada de la siguiente manera

$$F_{V_{ri}^t}(p) = FD(p) + FS_{V_{ri}^t}(p) + FG_{V_{ri}^t}(p) \tag{3.1}$$

Como vimos anteriormente, calcular la fuerza restringida a un conjunto ($F_V(p)$, fuerza F restringida a V) es calcular las fuerzas sociales y granulares de un peatón únicamente con los elementos del conjunto.

Un segundo algoritmo (ver algoritmo 2) es propuesto donde se aplican estos conceptos. En el paso 3 se calculan las fuerzas que actúan sobre cada peatón restringida al conjunto de vecinos V_{ri}^t

Algoritmo 2

1: **Para** $t = 1$, ..., t_f cada Δt **Hacer**
2: **Para Todo** peatón p **Hacer**
3: Calcular $F_{V_{ri}^t}(p)$
4: **Fin Para**
5: **Para Todo** peatón p **Hacer**
6: Actualizar la posición de p
7: **Fin Para**
8: **Fin Para**

Como el proceso de descubrir vecinos es costoso, se podría obtener una mejora en la performance si en lugar de calcularlo en cada iteración logramos hacerlo cada cierto número de iteraciones, denominado K. La idea principal es obtener el conjunto de peatones tal que incluya a todos los conjuntos de vecinos utilizados durante un rango de tiempo. Formalmente este conjunto es

$$V_{ri}^r(p) = \bigcup_{t \in r} V_{ri}^t(p) \tag{3.2}$$

donde $r = [t, t + K * \Delta t)$, que es el mínimo conjunto que contiene a $V_{ri}^t(p)$ $\forall t \in r$.

Este conjunto debe ser calculado en el instante t y ser utilizado para las siguientes K iteraciones. Esto es imposible de realizar ya que es una contradicción. Para calcular el conjunto de vecinos de un peatón en un instante de tiempo t' es necesario conocer las posiciones de todos los peatones para el instante anterior. Si t' es mayor que t, no se cuenta con esa información ya que es lo que se está calculando, y no puede ser utilizado. Es decir, utilizar el conjunto de vecinos para el rango r, para calcular las posiciones en el instante t, requiere tener las posiciones de los peatones para los instantes de tiempo t' con $t' \in [t, t + K * \Delta t)$, lo cual no es posible. Suponiendo que los peatones se mueven a una velocidad máxima $vmax$, la posición de un peatón para las siguientes K iteraciones puede ser estimada con un círculo de radio $vmax * K * \Delta t$ alrededor de la posición para el instante t. Dos peatones separados a una distancia δ desplazándose en direcciones opuestas para encontrarse tendrán que recorrer la mitad de la distancia cada uno. Por lo tanto, el conjunto de peatones $V_{vmax*K*\Delta t+ri}^t$ incluye todos los peatones vecinos para algún instante de tiempo en el rango r si ningún

otro peatón se mueve, y el conjunto $V^t_{vmax*K*\Delta t*2+ri}$ toma una distancia del doble que la anterior, teniendo en cuenta de esta forma el movimiento de sus vecinos. El conjunto dado por la expresión 3.2, el cual era necesario pero incalculable, puede ser aproximado por V^t_δ con $\delta = vmax * K * \Delta t * 2 + ri$. En la figura 3.1(a) se muestra el área utilizada para calcular el conjunto de vecinos V^r_{ri} calculado en forma exacta para una trayectoria determinada, mientras que en la figura 3.1(b) se muestra el área para calcular la aproximación de dicho conjunto.

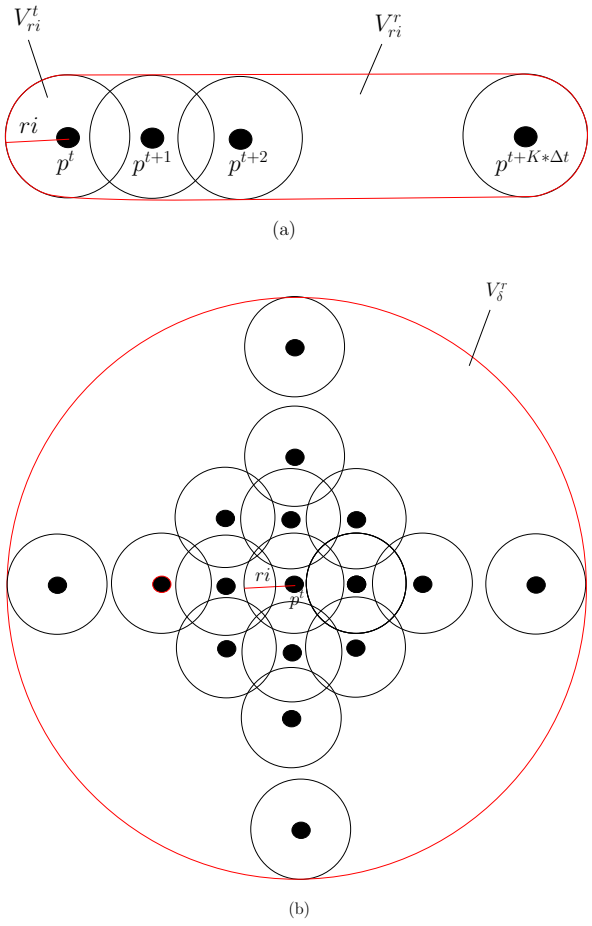


Figura 3.1: (a) Cálculo de los vecinos para un rango de tiempo donde el peatón se mueve en una dirección conocida(V^r_{ri}). (b) Aproximación de los vecinos para un rango de tiempo donde el peatón se mueve en todas las direcciones posibles (V^t_δ).

El algoritmo 3 es presentado como el algoritmo resultante del modelo computacional centralizado. Únicamente cada K iteraciones se calculan los conjuntos de vecinos para cada peatón y en el paso 7 se calculan las fuerzas que actúan sobre cada peatón restringidas al conjunto correspondiente.

Algoritmo 3

```
1: k ← 1
2: Para t = 1, ..., t_f cada Δt Hacer
3:    Para Todo peatón p Hacer
4:       Si k = 1 Entonces
5:          Calcular V ← V_δ^t(p)
6:       Fin Si
7:       Calcular F_V(p)
8:    Fin Para
9:    Para Todo peatón p Hacer
10:      Actualizar la posición de p
11:   Fin Para
12:   Si k = K Entonces
13:      k ← 1
14:   Sino
15:      k ← k + 1
16:   Fin Si
17: Fin Para
```

Calcular el K óptimo exige un compromiso entre el tiempo requerido para descubrir los vecinos y el cálculo de fuerzas con los mismos. Cuanto mayor sea K, mayor será V_δ^t y más fuerzas de interacción se calcularán. Mientras que la relación entre δ y K es lineal, el área del círculo determinado por δ crece en forma cuadrática, por lo que la cantidad de peatones que se encuentran en la misma crece también en esa proporción (en el caso de tener densidades altas), esquematizado en la figura 3.2(a).

Muchas técnicas han sido estudiadas para resolver eficientemente el problema de vecindad [18, 19] las cuales generalmente utilizan índices espaciales jerárquicos [16, 17]. Para hacer uso de los índices espaciales, antes de comenzar con el cálculo de vecinos de cada peatón para el instante t, todos los peatones son agregados al índice. Luego, calcular los $V_\delta^t(i)$ comprende consultar qué peatones se encuentran en el círculo de radio δ con centro en la posición del peatón i. Para obtener una performance aún mejor, para las consultas en los índices espaciales vistos, son convenientes las regiones rectangulares. Entonces, el círculo de radio δ es aproximado por el cuadrado de lado $2 * \delta$ como se ve en la figura 3.2(b).

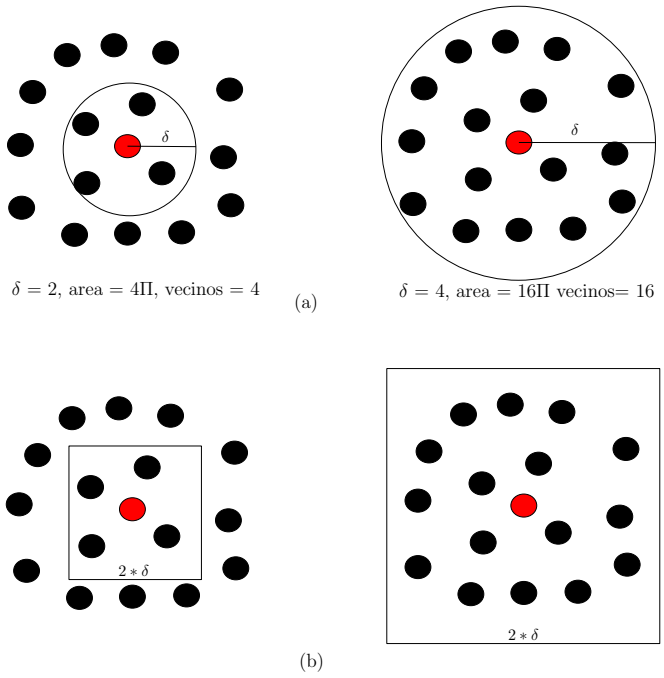


Figura 3.2: Cálculo de la vecindad de un peatón. (a) Usando un círculo de radio δ, (b) Aproximación por un cuadrado de lado $2 * \delta$.

3.2. Modelo Computacional Distribuido

Un modelo compuesto por miles de agentes requiere una gran cantidad de recursos para poder ser ejecutado en forma eficiente. En este trabajo utilizamos un cluster de computadoras como alternativa para la solución del problema de tráfico peatonal. Esto permite contar con muchos recursos computacionales y escalar en forma flexible y económica. En un modelo distribuido de agentes, estos residen en los distintos nodos del cluster e interactúan a través de mensajes. Distinguiremos *mensajes internos* de *mensajes externos*. Los primeros son los mensajes enviados entre agentes que residen en el mismo nodo, mientras que los segundos son los mensajes entre agentes de distintos nodos. Los mensajes externos pueden dar origen a comunicaciones entre los nodos correspondientes a cada agente. En este contexto, los costos de comunicación en el cluster son mucho más altos que el costo computacional [15]. Para obtener una buena performance de un sistema distribuido es necesario mantener balanceada la carga y minimizar las comunicaciones. Para lograr esto en un modelo de agentes distribuido es necesario distribuir los agentes manteniendo el balance de carga y minimizando los mensajes externos que se envían.

3.2.1. Probabilidad de Interacción

Llamamos *Probabilidad de Interacción* (PI) a la probabilidad de que dos agentes se envíen mensajes. Si los agentes residen en distintos nodos, mayor PI implica mayor probabilidad de comunicaciones. Por lo tanto, para disminuir las comunicaciones, debemos mantener bajo esta probabilidad entre agentes de distintos nodos. Una forma de lograr esto es haciendo que los pares de agentes que tengan una alta probabilidad de interacción tiendan a estar en el mismo nodo.

Consideramos dos formas de estimar el PI:

- Basados en el historial: la idea es estimar el PI en base a los mensajes registrados para un período de tiempo anterior. Suponemos que la probabilidad de mensajes futuros depende de los mensajes pasados.
- Basados en el estado: la idea es estimar el PI en base al estado de los agentes. Suponemos que la probabilidad de mensajes futuros depende del estado de los agentes y su entorno.

En el problema de tráfico peatonal los mensajes están determinados por las posiciones de los agentes. A una menor distancia entre agentes, mayor PI. Con el fin de minimizar las comunicaciones ubicaremos los peatones cercanos en el mismo nodo. En la figura 3.3 se ve una asignación de peatones en nodos según la distancia entre los mismos, donde el color representa a que nodo es asignado cada peatón.

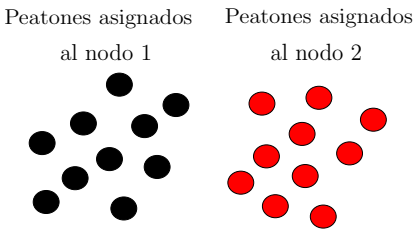


Figura 3.3: Con el criterio de que a menor distancia, mayor probabilidad de comunicación, los peatones cercanos entre sí son asignados al mismo nodo.

3.2.2. Balance de carga

El objetivo es encontrar una distribución de agentes en nodos de forma tal que la cantidad de cómputos que realiza cada procesador se encuentre balanceada. Con el fin de minimizar las comunicaciones, esta distribución debe estar sujeta a que los

peatones que se encuentran más cerca sean ubicados en el mismo nodo. En el caso de que no todos los procesadores sean iguales, el balance de carga debe tener en cuenta la diferencia de capacidades y la asignación debe ser proporcional a la misma. En nuestro problema, los cómputos realizados por cada procesador corresponden a:

1. Actualización de la posición del peatón

2. Cálculo de fuerzas de interacción

Mientras que los cómputos realizados en el punto 1 son constantes por cada peatón, los realizados en 2 dependen de la cantidad de vecinos con los que interactúa. Los nodos que tengan asignados conjuntos de peatones en los cuales haya un mayor número de interacciones, tendrán que realizar una mayor cantidad de cómputos. Por lo tanto para balancear la carga se tiene que tener en cuenta tanto la cantidad de peatones asignados a cada procesador, como la cantidad de interacciones entre los mismos. En la figura 3.4 se muestra un ejemplo donde ambos nodos tienen asignada la misma cantidad de peatones, pero en el nodo 1 es necesario calcular las interacciones entre peatones vecinos, mientras que en el 2 no. Por simplificación, en este trabajo consideramos situaciones donde la cantidad de interacciones por peatón es semajante, por lo que basta considerar la cantidad de peatones por nodo para lograr un balance de carga.

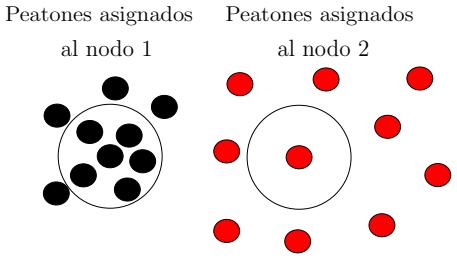


Figura 3.4: Ejemplo donde ambos nodos tienen asignada la misma cantidad de peatones, pero en el nodo 1 es necesario calcular las interacciones entre peatones vecinos, mientras que en el 2 no.

Para distribuir los peatones en los distintos nodos, primero particionamos el espacio en regiones (ver capítulo 4) y asignamos cada una a un nodo distinto. Luego, los peatones que se encuentran dentro de cada región son asignados al nodo correspondiente. Por ejemplo, para lograr la distribución de los peatones mostrada en la figura 3.3 se crean las regiones 1 y 2, como se muestra en la figura 3.5. Para este trabajo consideramos que el conjunto de obstáculos que componen la geometría de la planta donde se

esta realizando la simulación, no es muy grande por lo que es replicado en cada nodo del cluster.

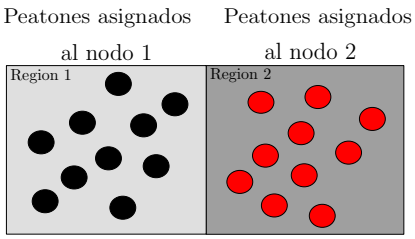


Figura 3.5: Creación de las regiones para lograr la distribución en dos nodos mostrada en la figura 3.3

A medida que los peatones cambian de posición, alguno podría salir de su región. En este caso es necesario reasignar el agente al nodo de la nueva región. A esta reasignación la llamamos *migración*, ya que el agente migra de un nodo a otro. Luego de varias migraciones podríamos encontrarnos en una situación de desbalance de carga, siendo necesario rebalancear. Para rebalancear es necesario recalcular las regiones, siendo deseable que las nuevas regiones sean similares a las anteriores con respecto a los peatones que contienen, para evitar la mayor cantidad de migraciones posible luego de este proceso. Este problema es conocido como *balanceo dinámico de la carga* y los métodos de distribución de agentes por regiones, que se verán en el capítulo 4, lo tienen en cuenta.

Migrar los peatones a otros procesadores produce un desbalance de la carga, pero permite mantener el invariante de que cada peatón reside en el nodo que tiene asociada la región que lo contiene. De esta forma, las consultas correspondientes al conjunto de peatones contenidos en una subregión se pueden realizar eficientemente al quedar restringuidas únicamente a un conjunto de procesadores (aquellos que tienen asociada una región que intersecta con la subregión de la consulta).

3.2.3. Algoritmo distribuido

Como vimos en el modelo centralizado, para un peatón definimos V_δ^t al conjunto de vecinos a distancia menor o igual que δ para el instante de tiempo t. Este conjunto contiene todos los peatones con los que va a interactuar el peatón en el rango de tiempo $[t, t + K * \Delta t)$. Como consecuencia de distribuir los agentes en nodos, dicho conjunto puede estar formado por peatones asignados al mismo nodo, llamados *vecinos internos* y por peatones asignados a otros, llamados *vecinos externos*. Si para un peatón p y

un nodo N notamos $V^t_{\delta,N}(p) = \{\ v\ \textit{peatón} : v \in V^t_\delta(p) \wedge v \in N\ \}$ tendremos que $V^t_\delta(p) = \bigcup_{n:Nodo} V^t_{\delta,n}(p)$. Si notamos N_p al nodo donde esta asignado el peatón p, podemos reescribir $V^t_\delta(p)$ de la siguiente forma

$$V^t_\delta(p) = V^t_{\delta,N_p}(p) \ \cup \bigcup_{n \neq N_p} V^t_{\delta,n}(p) \tag{3.3}$$

Nótese que el primer término de esta expresión corresponde a los vecinos internos y el segundo a los externos.

Sea N un nodo, definimos $\Psi(N)$ al conjunto de peatones asignados al nodo N $\Psi(N) = \{\ p\ \textit{peatón} : p \in N\ \}$ y $V^t_\delta(N)$ al conjunto de vecinos de los peatones asignados a N $V^t_\delta(N) = \bigcup_{p \in N} V^t_\delta(p)$.

Para que un procesador realice los cálculos correspondientes para sus peatones debe conocer las posiciones y atributos de los peatones pertenecientes a $V^t_\delta(N)$. Esto comprende tanto vecinos internos como externos. Los vecinos internos residen en el mismo nodo, en cambio los vecinos externos deben ser requeridos a los nodos correspondientes. Para realizar el cálculo de posiciones eficientemente, cada procesador deberá notificar a los demás procesadores los peatones que cada uno necesita para obtener su conjunto de vecinos. Si N y M son dos nodos, N notificará a M $\Psi(N) \cap V^t_\delta(M)$ (que son los peatones asignados a N y que están en el conjunto de vecinos de los peatones asignados a M) y, en forma análoga, M notificará a N $\Psi(M) \cap V^t_\delta(N)$. Notaremos $\mathcal{N}(N, M)$ a los peatones que N notificará a M

$$\mathcal{N}^t_\delta(N, M) = \Psi(N) \cap V^t_\delta(M) \tag{3.4}$$

Para calcular el conjunto de peatones $\mathcal{N}^t_\delta(N, M)$, se define la *región de notificación* $\mathbb{N}(N, M)$ entre los nodos N y M, como el área que lo contiene. Para calcular el área de notificación definimos el *buffer* de un área A para una distancia d, $Bfr_d(A)$, como la ampliación de A en d. Formalmente, asumiendo que estamos en $\mathbb{R}^2$, $Bfr_d(A) = \{\ x \in \mathbb{R}^2 : \exists y \in A : ||x - y|| \leq d\ \}$. La región de notificación puede ser calculada con la siguiente expresión

$$\mathbb{N}_\delta(N, M) = \textit{región}(N) \cap Bfr_\delta(\textit{región}(M)) \tag{3.5}$$

donde *región*(N) representa la región asignada al nodo N. Para simplificar la notación nos referiremos al nodo N y a la *región*(N) simplemente como N, entendiéndose

por el contexto a cual nos estamos referiendo. De esta forma se puede reescribir la ecuación 3.5 de la siguiente forma

$$\mathbb{N}_\delta(N, M) = N \cap Bfr_\delta(M) \tag{3.6}$$

En la figura 3.6 se ve un ejemplo del cálculo de la región $\mathbb{N}(N, M)$. En (a) se muestran las regiones originales, en (b) la región de M ampliada y en (c) el resultado de la intersección. En la figura 3.6(b) se muestra un peatón asignado al nodo M, el cual se encuentra en el borde de la región, y por ende, el conjunto de vecinos contendrá peatones asignados al nodo N.

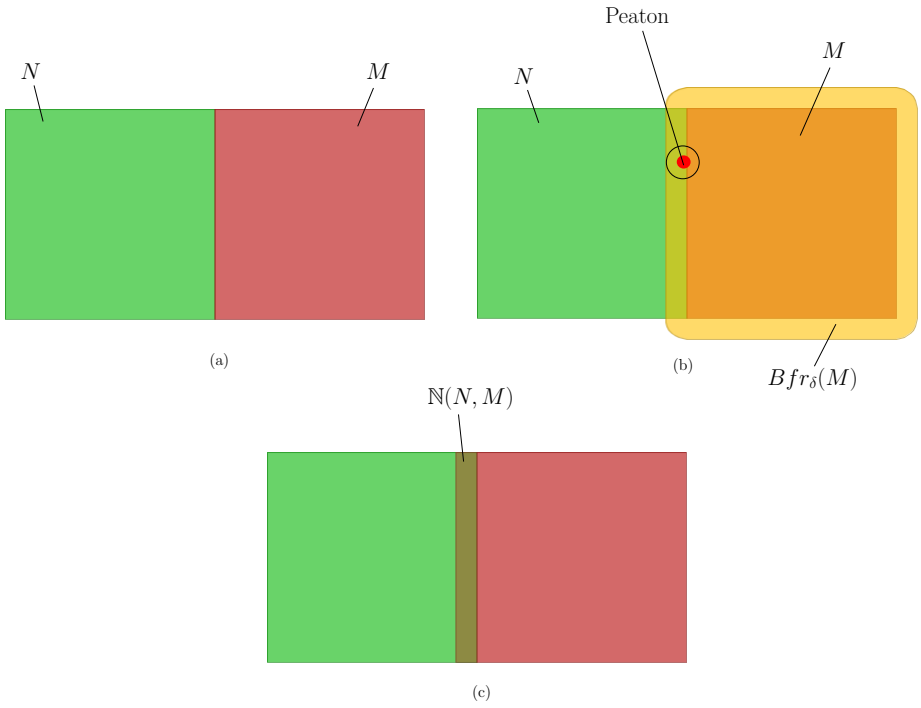


Figura 3.6: Ejemplo del cálculo de la región de notificación $\mathbb{N}(N, M)$, en (a) se muestran las regiones originales, en (b) la región de M ampliada y en (c) el resultado de la intersección.

Como dijimos anteriormente, los peatones que al moverse salgan de la región del procesador de donde están asignados, deben ser migrados. El conjunto de peatones formado por los peatones del nodo N que se encuentren contenidos dentro de la región de M $\forall M \neq N$ para el instante de tiempo t, son los que deben ser migrados al nodo M y los llamaremos $\mathcal{M}^t(N, M)$. Como la región de M puede ser muy compleja, calcular los peatones contenidos dentro de la misma puede demandar muchos cálculos (ver la

siguiente sección). Para esto, se define la *región de migración* $\mathbb{M}(N, M)$ entre los nodos N y M con la siguiente expresión

$$\mathbb{M}(N, M) = Bfr_{\tau}(N) \cap M \tag{3.7}$$

donde τ es un parámetro que determina el tamaño de la región resultante. En la figura 3.7 se ve un ejemplo del cálculo de la región de migración $\mathbb{M}(N, M)$. En (a) se muestran las regiones originales, en (b) se agrega la región correspondiente al primer término de la expresión 3.7, y en (c) el resultado de la intersección con la región de M. Como los peatones tienen un límite máximo de velocidad, si se define τ de la forma correcta, los peatones que ingresan a la región del nodo M provenientes de N deben primero pasar necesariamente por la región de migración. El cálculo de los peatones a migrar se realiza con la misma frecuencia que la del descubrimiento de vecinos, $K * \Delta t$, por lo que definimos $\tau = K * \Delta t * vmax$ de forma que se cumpla lo anterior.

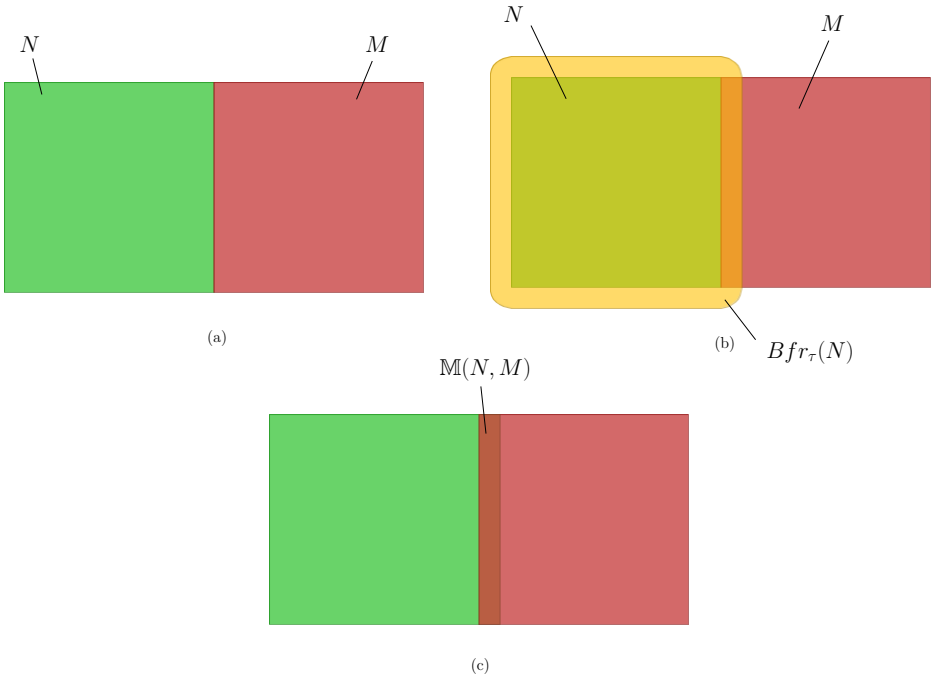


Figura 3.7: Ejemplo del cálculo de la región de migración $\mathbb{M}(N, M)$, en (a) se muestran las regiones originales, en (b) se agrega la región correspondiente al primer término de la ecuación 3.7, y en (c) el resultado de la intersección con la región de M.

Definimos *nodos vecinos* como los pares de nodos que tienen una región de notificación o migración no vacía, definiendo así el conjunto $V(N) = \{\ M : \mathbb{N}(N, M) \neq \emptyset \vee \mathbb{M}(N, M) \neq \emptyset\ \}$. Sólo se realizan notificaciones y/o migraciones con los nodos ve-

cinos, de esta forma se reduce la comunicación entre procesadores. Esto es importante ya que la cantidad de comunicaciones no se incrementa en forma cuadrática al agregar más procesadores para resolver un problema.

A continuación, presentamos el algoritmo distribuido (ver algoritmo 4), el cual parte de modificar el algoritmo 3 introducido en la versión centralizada (sección 3.1).

Algoritmo 4

Entrada: N nodo donde se ejecuta el algoritmo

1: $k \leftarrow 1$
2: **Para** $t = 1$, ..., t_f cada Δt **Hacer**
3: **Si** $k = 1$ **Entonces**
4: **Para Todo** nodo $M \in V(N)$ **Hacer**
5: Calcular $\mathcal{N} \leftarrow \mathcal{N}_\delta^t(N, M)$
6: Notificar a M el conjunto de peatones $\mathcal{N}$
7: Calcular $\mathcal{M} \leftarrow \mathcal{M}^t(N, M)$
8: Migrar a M el conjunto de peatones $\mathcal{M}$
9: **Fin Para**
10: **Sino**
11: **Para Todo** procesador $M \in V(N)$ **Hacer**
12: Notificar a M el conjunto de peatones $\mathcal{N}$
13: **Fin Para**
14: **Fin Si**
15: Esperar que lleguen todas las notificaciones de $V(p)$
16: **Para Todo** peatón p **Hacer**
17: **Si** $k = 1$ **Entonces**
18: Calcular $V \leftarrow V_\delta^t(p)$
19: **Fin Si**
20: Calcular $F_V(p)$
21: **Fin Para**
22: **Para Todo** peatón p **Hacer**
23: Actualizar la posición de p
24: **Fin Para**
25: **Si** $k = K$ **Entonces**
26: $k \leftarrow 1$
27: **Sino**
28: $k \leftarrow k + 1$
29: **Fin Si**
30: **Fin Para**

Con estas modificaciones al algoritmo 3, cada procesador puede calcular las fuerzas de interacción de todos los peatones que tiene asignado y actualizar sus posiciones. En el paso 15 se espera a que lleguen todas las notificaciones de los procesadores vecinos

para asegurarse que cuenta con toda la información necesaria. El tiempo ocioso del procesador durante esta espera puede ser aprovechado para calcular todas las fuerzas de interacción con vecinos internos sin esperar los datos de los otros procesadores. En las iteraciones donde se calculan los conjuntos de vecinos (paso 18), cuando se recibe una notificación, se agregan los peatones a las vecindades de los peatones locales ya calculadas.

3.2.4. Cálculo de peatones a notificar y a migrar

Las regiones de notificación $\mathbb{N}(N, M)$ y migración $\mathbb{M}(N, M)$, pueden ser calculadas por única vez luego de ser asignadas las regiones a cada procesador. El conjunto de peatones contenidos dentro de una región se puede definir como todo peatón p cuya distancia a la región es menor que el radio del peatón, considerando la distancia entre un peatón y una región como la distancia euclidiana de la posición del peatón al punto más cercano de la región. En la figura 3.8 se ve un ejemplo de un conjunto de peatones contenidos dentro de una región representada por un polígono.

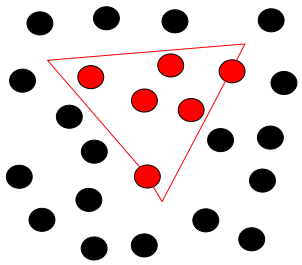


Figura 3.8: Cálculo de peatones contenidos en una región representada por un polígono.

Para calcular eficientemente los peatones contenidos en una región utilizamos el *bounds* de la región, el cual definimos como el rectángulo más pequeño que contiene totalmente a la misma. El procedimiento para el cálculo consta de los siguientes pasos:

1. Calcular cuáles peatones están contenidos dentro del bounds de la región.

2. Verificar cuáles de los peatones resultantes del paso anterior pertenecen efectivamente a la región misma.

Para el paso 1 utilizamos un índice espacial implementado sobre un *QuadTree* [16, 17], donde las consultas por rectángulos son muy eficientes. En la figura 3.9 se ve un ejemplo de este procedimiento.

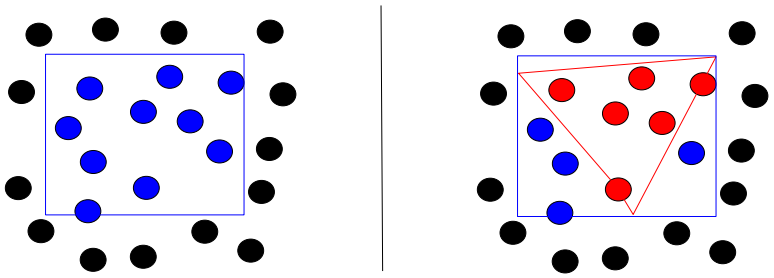


Figura 3.9: Procedimiento para calcular los peatones contenidos en una región utilizando el bounds de la región. Los puntos azules representan a los peatones contenidos en el bounds de la región pero no en la región misma.

Sin embargo, este procedimiento puede tener casos muy malos como ser el que se ve en la figura 3.10(a). Al filtrar los peatones en el paso 1 no se produce ninguna reducción en el conjunto de peatones con los cuales se realiza el paso 2. Para solucionar este problema, en vez de utilizar el bounds de la región, se utiliza una aproximación de ésta calculada por rectángulos en forma recursiva. Diremos que una región r^* aproxima bien a una región r si el error de aproximación es menor que un ϵ, definiendo el error como $area(r \cap r^*)/area(r)$. Si el bounds de una región no es una buena aproximación, este se divide en cuatro cuadrantes, y se realiza el mismo procedimiento con cada subregión resultante de la intersección de cada cuadrante con la región misma. En la figura 3.11 se ve un ejemplo de esta aproximación, mientras que en la figura 3.10(b) se ve un ejemplo de como se utilizaría esto en el cálculo de peatones dentro de una región.

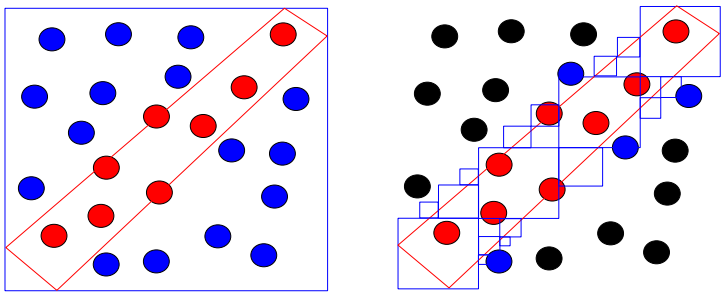


Figura 3.10: (a) Ejemplo de un caso donde el rectángulo que contiene al polígono no sirve como primera aproximación para filtrar el conjunto total de puntos. (b) Solución al problema anterior utilizando una aproximación por rectángulos de la región original

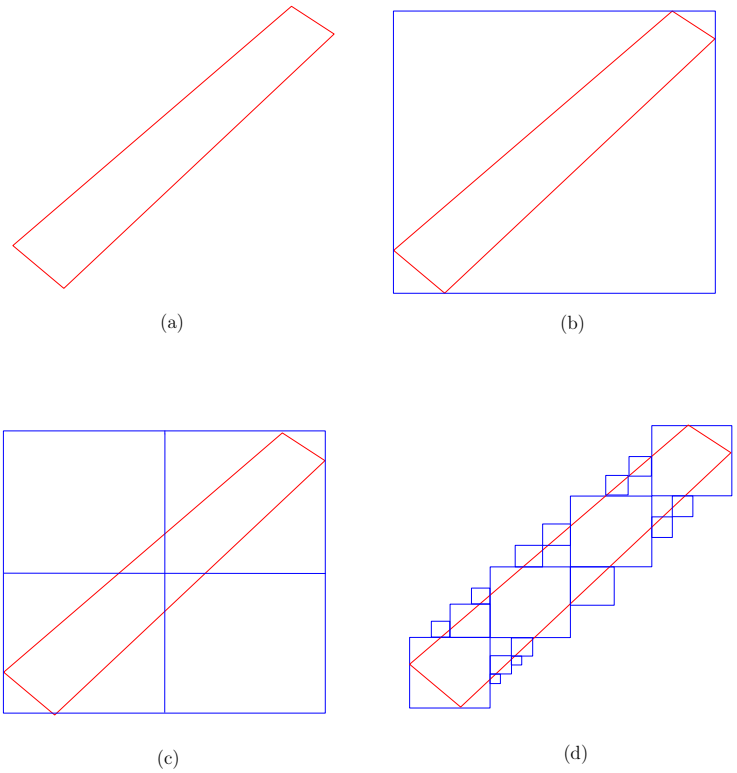


Figura 3.11: Ejemplo de aproximación de una región por rectángulos. (a) Región original. (b) Primer paso, aproximación por el bounds de la región. (c) Paso recursivo, se divide el rectángulo en cuatro. (d) Resultado.

Capítulo 4

Partición del Espacio

[1]Como vimos en el capítulo anterior, el espacio es particionado en regiones para lograr una distribución de los peatones en procesadores, de forma tal que lo peatones cercanos sean ubicados en el mismo nodo. Para lograr esta partición, utilizamos un método basado en grafos el cual permite modelar las relaciones entre los peatones. El método de partición del espacio para la creación de regiones consiste en:

1. Obtener las posiciones de los peatones
2. Crear el grafo de interacciones
3. Particionar el grafo
4. Crear las regiones

4.1. Obtener las posiciones de los peatones

En este trabajo utilizamos un algoritmo de partición centralizado, que requiere conocer la ubicación de todos los peatones del sistema. Esta información se debe recolectar de los distintos procesadores, la cual es necesaria para la creación y partición del grafo. Si bien existen algoritmos paralelos, un algoritmo centralizado es suficientemente bueno para los tamaños de problemas analizados en este trabajo [4].

[1]Recomendamos leer el apéndice A antes de continuar con este capítulo.

4.2. Crear el grafo de interacciones

El segundo paso consiste en construir un grafo de interacciones. El mismo esta formado por vértices que representan a los peatones, y ejes que representan las interacciones. El conjunto de vértices esta claramente definido al asociar cada vértice con un peatón. Nos referiremos a vértice y peatón en forma indistinta.

Dado que los algoritmos de partición consideran los ejes y sus pesos, que representan la "fuerza del vinculo" entre los nodos, distintas asignaciones de pesos redundan en distintas particiones. Una posibilidad es crear un grafo completo asignando los pesos de los ejes de forma tal que en el paso siguiente, al calcular la partición del mismo, se obtenga el resultado esperado: ubicar los peatones cercanos en el mismo subdominio. El problema de este método, es que la cantidad de ejes de un grafo completo crece con orden $O(n^2)$ siendo n la cantidad de nodos. Para reducir la cantidad de ejes del grafo, no se agregan los ejes que unen peatones cuya distancia es mayor que un cierto μ (la distancia entre los peatones unidos por el eje determina la longitud del mismo). Esto disminuye significativamente la cantidad de ejes a agregar, pero sin embargo, en las zonas donde existe una alta densidad de peatones todavía se contará con una gran cantidad de ejes. Por otro lado, los resultados de este método son muy sensibles al valor de μ, ya que elegir un valor muy chico permite reducir más la cantidad de ejes pero produce muchas componentes disconexas, y elegir un valor muy grande produce el efecto contrario. El valor de μ esta relacionado con la densidad de peatones en el momento de crear el grafo, y determina cual es la distancia máxima a la que se conectan los vértices. Este es un parámetro independiente del valor de δ utilizado para el descubrimiento de vecinos. En la figura 4.1 se ve un ejemplo de la creación de un grafo con este método con distintos valores del parámetro μ.

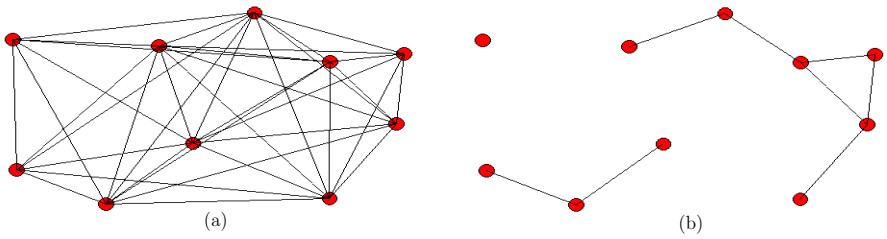


Figura 4.1: Ejemplo de creación del grafo completo con distintos valores del parámetro μ en un problema con una densidad aproximadamente de 1 peatón por metro cuadrado. (a) Ejemplo de un valor alto (mayor que 1). (b) Ejemplo de un valor bajo (menor que 1).

Para obtener un grafo con un método que no sea tan dependiente de la distribución de peatones se propone utilizar una triangulación entre los vértices (por ejemplo la *Triangulación de Delaunay*[27]). De esta forma, la cantidad de ejes esta limitada por la cantidad de ejes de un grafo planar: $3n - 6$, sin importar la distribución y densidad de los peatones. También es posible filtrar los ejes cuya distancia sea menor que μ, pero en este caso, una elección de un μ mas grande no trae como desventaja la gran cantidad de ejes en el grafo.

En la figura 4.2 se ve un ejemplo de creación del grafo con los métodos anteriores. En (a) se muestra el grafo completo y en (b) el grafo sin los ejes de longitud -mayor que μ. En el caso (b) se ve claramente como en la componente izquierda del grafo existe una gran cantidad de ejes por el alto valor de μ. Sin embargo, este alto valor no es suficiente para crear un grafo conexo. En (c) se muestra el grafo creado a partir de la triangulación de Delaunay y en (d) el mismo grafo sin los ejes de longitud mayor que μ. En este último caso, se pudo usar un valor de μ mayor y no generar componentes disconexas.

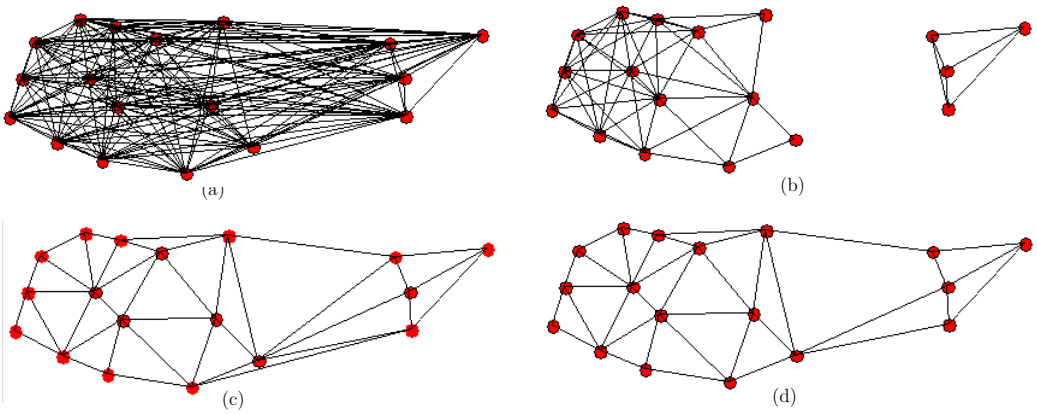


Figura 4.2: Creación del grafo a partir de las posiciones de los peatones. (a) Grafo completo. (b) Grafo (a) sin ejes de longitud mayor que μ. (c) Grafo a partir de la triangulación de Delaunay. (d) Grafo (c) sin ejes de longitud mayor que μ.

Peso de los ejes: La función utilizada para la asignación de los pesos es la siguiente:

$$w = a * e^{-d^2/b} \tag{4.1}$$

donde d es la distancia entre los vértices, a controla el valor máximo de la curva y b su decaimiento. Esta función permite asignarle un valor alto a los vértices mas cercanos

y disminuir el valor suavemente tendiendo a cero a medida que la distancia aumenta. El valor de μ utilizado es aquel que hace que el peso asignado a un eje de esa longitud sea menor que un cierto ϵ.

Componentes disconexas: Las componentes disconexas pueden generar resultados indeseables e, inclusive, algunos algoritmos de partición no aceptan grafos de estas características. Para conectar las distintas componentes disconexas resultantes del algoritmo de creación del grafo, se crea un nuevo grafo tomando como vértice cada una de las componentes creando una nueva triangulación entre esos vértices y luego conectando cualquier vértice del grafo original que representa cada componente por cada segmento de la triangulación.

En la figura 4.3 se muestra un ejemplo de la creación de un grafo utilizando triangulación para agregar los ejes y para conectar las componentes disconexas.

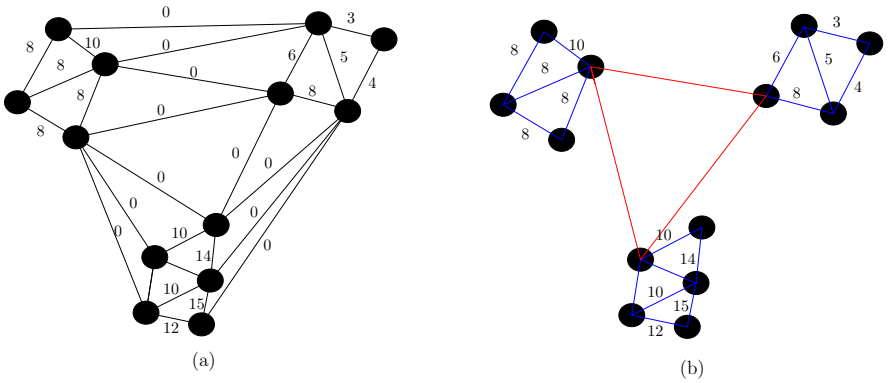


Figura 4.3: Creación de un grafo utilizando triangulación para agregar los ejes (a) Agregando todos los ejes de la triangulación (b) Filtrando los ejes de peso 0 y conectando las componentes disconexas.

4.3. Partición del grafo

En esta etapa del método de partición del espacio es necesario obtener una partición del grafo planteado previamente. Para realizar esto se utiliza un algoritmo de bisección recursiva multinivel [4] para partición estática de grafos (ver apéndice A). Un algoritmo de partición de grafos consiste en agrupar los vértices en subdominios de forma que cada subdominio tenga la misma cantidad de vértices y se minimice la cantidad de ejes entre subdominios (conocidos como *edge-cut*).

Sin embargo, este algoritmo no tiene en cuenta la dinámica de los cambios en el grafo. Para minimizar las migraciones entre procesadores luego de obtener una nueva partición se aplican dos técnicas distintas basadas en *partición desde cero* (ver apéndice A), una antes particionar y otra al finalizar. La modificación al grafo antes de particionar consiste en crear un nodo virtual por cada procesador y conectar cada vértice del grafo con el nodo virtual correspondiente al procesador que tiene asignado el peatón. De esta forma, migrar un nodo tendría su costo de penalización al incluir este eje en el edge-cut. La otra modificación al algoritmo consiste en un análisis postpartición para reetiquetar los números de partición asignados a cada subdominio obtenido, de forma que se minimicen las migraciones. Se renombran las etiquetas de los subdominios de forma tal que se mantenga la mayor cantidad de vértices en común con los subdominios anteriores.

En la figura 4.4 se muestra la partición del grafo creado en la figura 4.3(b), en (a) una partición inicial y en (b) una nueva partición minimizando las migraciones luego de que los peatones se movieran.

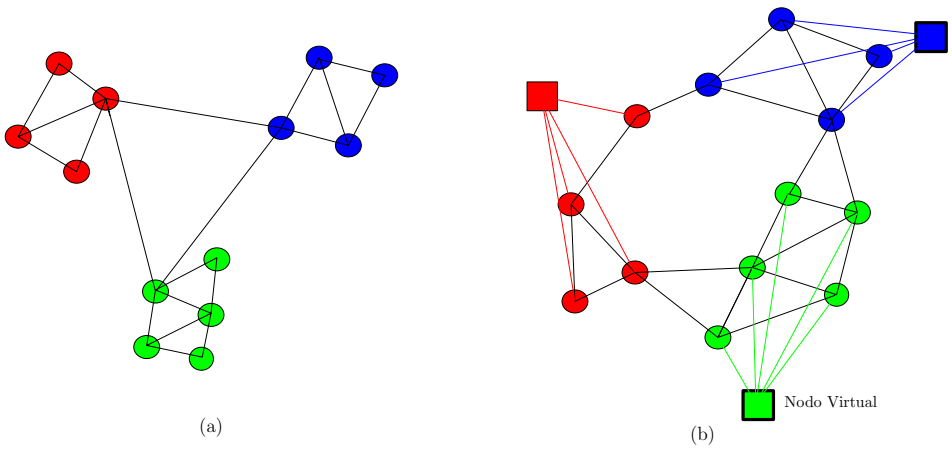


Figura 4.4: Partición del grafo de la figura 4.3(b) en tres subdominios. (a) Partición inicial. (b) Nueva partición minimizando las migraciones luego de que los peatones se movieran.

4.4. Creación de regiones

Una vez obtenida la partición del grafo es necesario calcular las regiones que determinan la partición del espacio. Para realizar esto se utiliza como base el *Diagrama de Voronoi* [27], el cual subdivide el espacio en regiones, llamadas *celdas*, de acuerdo a

cuál es el punto característico más cercano, ver figura 4.5(a).

Se toma el conjunto de vértices del grafo como el conjunto de puntos característicos del Diagrama de Voronoi. A cada celda resultante se le asigna la partición del vértice que contiene. Las celdas adyacentes que tienen asociadas la misma partición son unidas de forma tal de crear polígonos más grandes, los cuales forman las regiones. En la figura 4.5(b) se ve el diagrama de Voronoi con la asignación de particiones a las celdas donde cada partición es representada por un color y en 4.5(c) se ve el diagrama de regiones donde las celdas adyancentes son unidas.

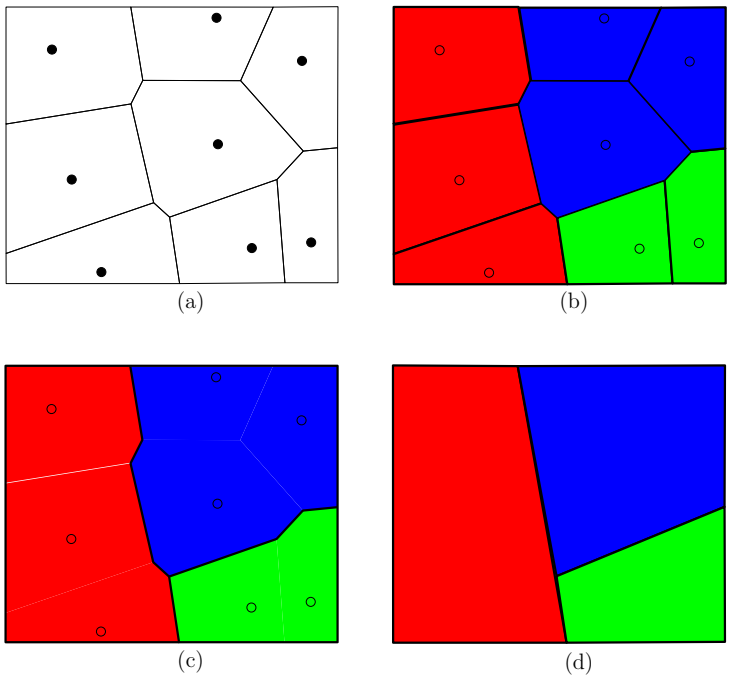


Figura 4.5: Ejemplo del diagrama de Voronoi de un conjunto de 8 puntos (a) diagrama clásico, (b) diagrama con la asignación de particiones a las celdas, (c) diagrama de regiones, (d) diagrama de regiones simplificado.

Para realizar esto en forma eficiente, en vez de calcular la unión de las celdas adyacentes se modificó el algoritmo que genera el diagrama de Voronoi. Cada segmento que compone el diagrama separa dos puntos del conjunto de puntos característicos. Entonces, sólo se agregan los segmentos que separan a vértices de diferentes particiones. De esta forma, las celdas obtenidas por el diagrama son las resultantes de realizar la unión entre las correspondientes. El Diagrama de Voronoi puede ser obtenido a partir de la triangulación de Delaunay, la cual fue calculada en el momento de crear el grafo al comenzar con el algoritmo de partición.

Para crear las regiones que determinan la partición del espacio, aplicamos el algoritmo 5, el cual toma como entrada los segmentos correspondientes al Diagrama de Voronoi. Las regiones obtenidas a partir del algoritmo presentado, no representan exactamente las regiones obtenidas por el Diagrama de Voronoi, sino que estas son simplificadas para obtener polígonos de una menor cantidad de puntos (ver figura 4.5(d)). Como las operaciones que se realizan con las regiones tienen un orden de complejidad dependiente de la cantidad de puntos que tengan las polígonos que determinan las mismas, simplificar las regiones permite reducir los costos en el momento de ser usadas.

Algoritmo 5 Creación de regiones

Entrada: C conjunto de segmentos
Salida: Conjunto de polígonos que determinan las regiones

1: $M \leftarrow$ lineMerger(C) //conecta un conjunto de segmentos formando lineas mas largas
2: **Para Todo** linea $l \in M$ **Hacer**
3: $l \leftarrow$ simplify(l) //simplifica una linea por otra que la aproxime utilizando una menor cantidad puntos
4: **Fin Para**
5: $P \leftarrow$ polygonize(M) //crea poligonos a partir de lineas
6: **Para Todo** polígono $p \in P$ **Hacer**
7: $pto \leftarrow$ puntoInterior(p) //punto interior cualquiera dentro del poligono
8: Asignar a p el número de partición del peatón mas cercano a pto
9: **Fin Para**
10: **Devolver** P

En la figura 4.6 se ven las regiones correspondiente a los grafos particionados en la figura 4.4.

4.5. Ejemplo de una corrida

Para ejemplificar como las regiones se adaptan a los movimientos de los peatones, en la figura 4.7 se muestran los cambios de regiones a lo largo del tiempo de una simulación de evacuación de una habitación.

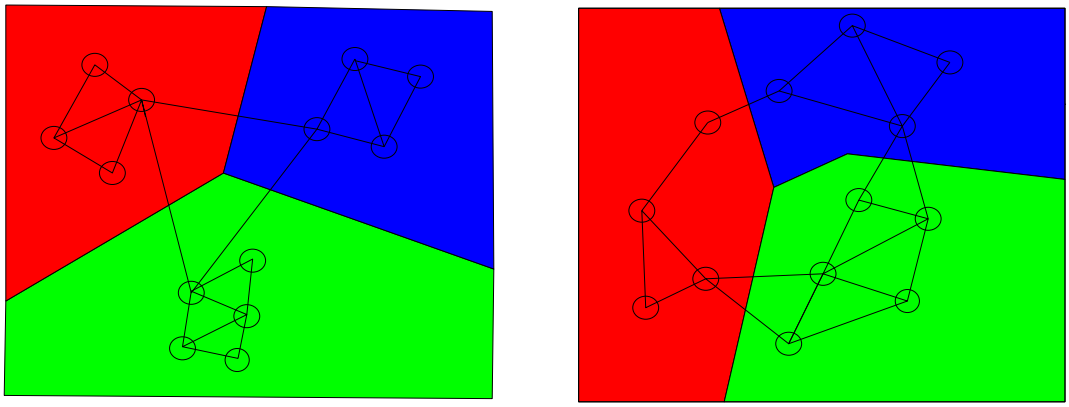


Figura 4.6: Creación del diagrama de regiones a partir de los grafos presentados en las figura 4.4(a) y 4.4(b)

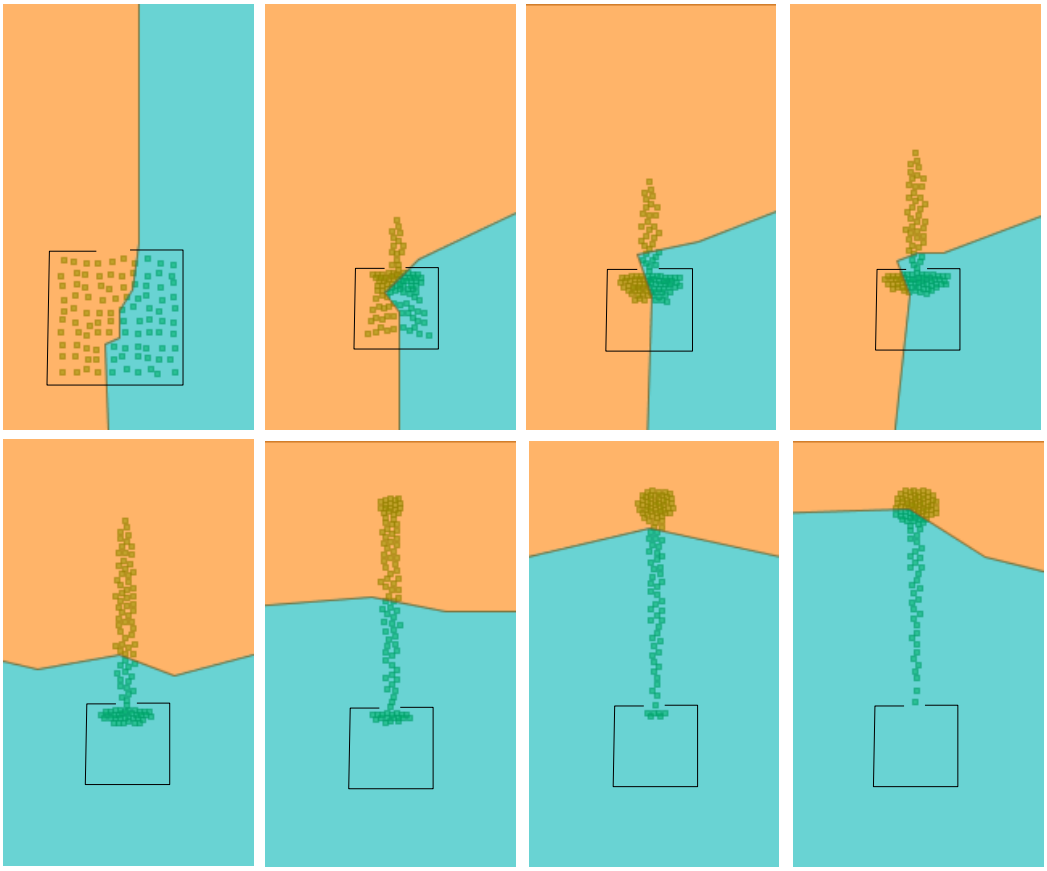


Figura 4.7: Ejemplo de una simulación de evacuación de una habitación.

Capítulo 5

Simulaciones y Resultados

5.1. Implementación

La implementación del simulador de tráfico peatonal distribuido esta desarrollada en el lenguaje de programación *JAVA 5.0*. Utilizamos el framework de simulación basada en agentes *Distributed Agents* desarrollado por la empresa *Urbix Technologies S.A.*[1], el cual permite crear un modelo de agentes facilitando el envío de mensajes entre los mismos. Para realizar las comunicaciones entre los distintos procesadores del cluster utiliza la tecnología RMI[2].

Para las operaciones geométricas, como ser la implementación de quadtree y manipulación de polígonos se utilizó la librería *JTS Topology Suite* desarrollada por la empresa *Vividsolutions*[3].

Para calcular la triangulación de Delaunay se utilizó la libreria *Triangle* desarrollada por *Jonathan Richard Shewchuk*[4]. La misma esta implementada en C, por lo que se debió realizar una interfaz con JNI para poder ser llamada desde JAVA.

Para obtener una partición del grafo se utilizó la libreria *Metis* desarrollada por George Karypis, Vipin Kumar y Kirk Schloegel[5]. La misma cuenta con varias implementaciones de partición de grafos también desarrolladas en C, por lo que también se debió realizar una interfaz para poder ser llamadas desde JAVA. Se utilizó el método *METIS_PartGraphRecursive* basado en bisección recursiva multinivel [4, 11].

[1] http://www.urbix.com.ar

[2] http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp

[3] http://www.vividsolutions.com/jts/jtshome.htm

[4] http://www.cs.cmu.edu/ quake/triangle.html, Computer Science Division, University of California at Berkeley

[5] http://glaros.dtc.umn.edu/gkhome/views/metis, Karypis Lab

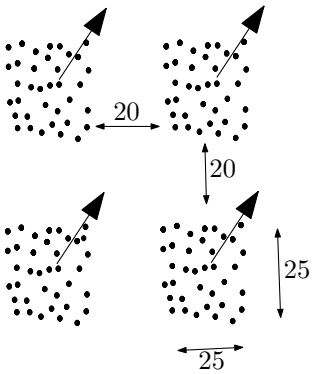
5.2. Equipamiento Utilizado

Las simulaciones realizadas para probar la implementación (descriptas en la siguiente sección) fueron realizadas en el Laboratorio de Imágenes del Departamento de Computación de la Facultad de Ciencias Exactas y Naturales. Cuenta con 8 máquinas Pentium IV 2.4 Ghz con 1 GB de RAM y conectadas con un switch Compaq Dual Speed 3322.

5.3. Simulaciones

Para evaluar la implementación del simulador de tráfico peatonal desarrollada en este trabajo de tesis se plantearon los siguientes dos problemas:

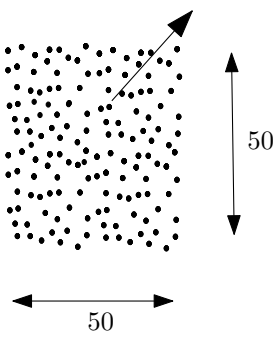
Problema A



Parametros:
Separacion entre los bloques: 20 m
Tamano del bloque: 25 m
Cantidad de peatones: 500 por bloque

Area del bloque: 625 m2
Area total cubierta: 2500 m2
Cantidad total de peatones: 2000
Densidad inicial: 1,25 peatones x metro

Problema B



Parametros:
Tamano del bloque: 50 m
Cantidad de peatones: 2000

Area del bloque: 2500 m2
Area total cubierta: 2500 m2
Cantidad total de peatones: 2000
Densidad inicial: 1,25 peatones x metro

Parámetros de ambos problemas:

Tiempo de simulación: 30 segundos
Parámetro A, función de asignación de pesos: 10
Parámetro B, función de asignación de pesos: 100
Peso del eje que conecta con el nodo virtual: 1
Δt: 0,01 segundos
K: 10
ri: 1.5 m
Masa del Peatón: 74
Radio del Peatón: 0.255 m
Velocidad Deseada: 2 m/s
Velocidad Máxima: 4 m/s

Todos los peatones tienen el mismo comportamiento y ejecutan el mismo plan en ambos problemas: se mueven en diagonal hacia arriba y la derecha. El movimiento es diagonal ya que se vió que las regiones calculadas en forma automática para distribuir los peatones en procesadores podían ser rectángulos longitudinales o transversales (ver figura 5.1), entonces un movimiento horizontal o vertical haría que en algunos casos no ocurra un desbalanceo. De esta forma el movimiento diagonal, por el contrario, hace que el dominio de peatones tienda a generar un desbalance de carga. Esto último, permite probar como es el desempeño del simulador en un problema de estas características donde es necesario adaptar las regiones que generan la distribución del problema, permitiendo analizar el balanceo dinámico de la carga frente al balanceo estático, el cual se efectúa por única vez al comenzar la simulación.

En la figura 5.2 y 5.3 se muestran los distintos estados en los que se encuentra la simulación del Problema A a lo largo del tiempo para un cluster de 2 nodos, utilizando balanceo estático de la carga y balanceo dinámico de la carga, respectivamente. Los puntos negros representan los peatones, y las zonas rojas y verdes representan las regiones asignadas a cada uno de los nodos. Comparando las situaciones anteriores se puede ver como en el segundo caso (utilizando balanceo dinámico de la carga), las regiones se adaptan al movimiento de los peatones manteniendo el sistema balanceado.

En la figura 5.4 se muestra el estado del sistema a lo largo del tiempo para el problema B. En este caso se ve cómo las regiones también se adaptan al movimiento de las personas.

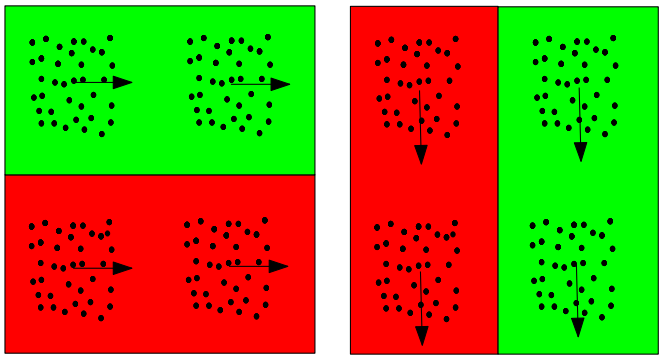


Figura 5.1: (a) Desplazamiento horizontal con rectángulos longitudinales, (b) Movimiento vertical con rectángulos transversales. En ninguno de los dos casos se produce desbalanceo.

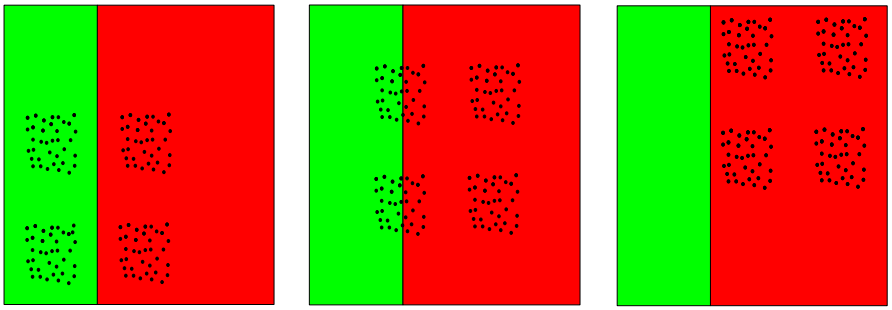


Figura 5.2: Problema A con balanceo estático de la carga para los instantes de tiempo 0 (izquierda), 15 (centro) y 30 (derencha) segundos.

Como el problema se desbalancea con la evolución en el tiempo, se analizó como deben ajustarse las regiones al movimiento de los peatones. En el caso del problema A, cómo la separación entre los bloques de personas es de 20 metros, el borde de la región calculada estará a 10 metros de los extremos de cada bloque. Como los peatones se mueven a 2 m/s, en menos de 5 segundos no podrán alcanzar dicho borde. Considerando este hecho, se decidió rebalancear cada 5 segundos para adaptar las regiones al movimiento de los peatones.

En el caso del problema B, como existe un único bloque de personas, inmediatamente luego de comenzar la simulación se contará con peatones en los bordes de las regiones, y luego, con peatones fuera de la misma provocando un desbalanceo. Igualmente para este problema se realizó una repartición cada 5 segundos de simulación.

Para ambos problemas se realizaron pruebas con distintas cantidades de procesadores, distintos tamaños de problema y con balanceo estático y dinámico de la carga.

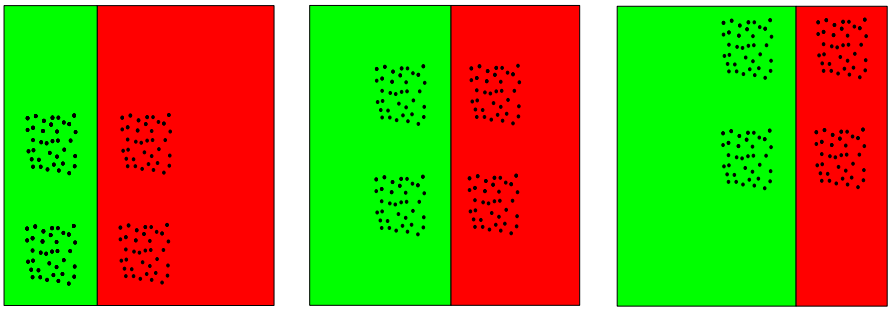


Figura 5.3: Problema A con balanceo dinámico de la carga para los instantes de tiempo 0 (izquierda), 15 (centro) y 30 (derencha) segundos.

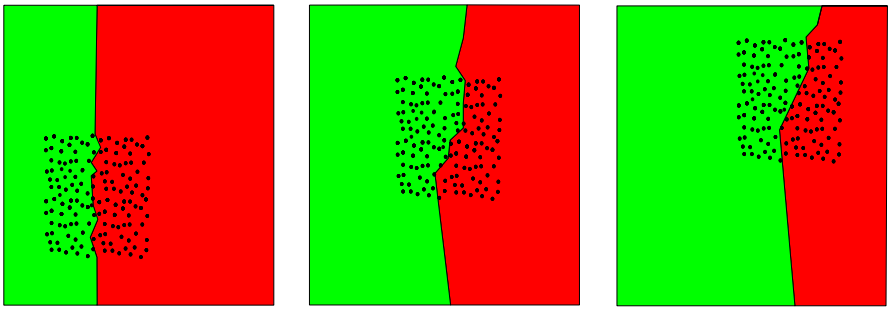


Figura 5.4: Problema B con balanceo dinámico de la carga para los instantes de tiempo 0 (izquierda), 15 (centro) y 30 (derencha) segundos.

Notaremos pP a una simulación en P procesadores (por ejemplo: $p2$, $p4$), y xX a un problema de tamaño X (por ejemplo: $x1$, $x10$), donde agrandar el problema por X significa multiplicar el tamaño del bloque por $\sqrt{X}$, la cantidad de peatones por X y dejar fija la separación entre los bloques. De esta forma, se mantiene constante la densidad de peatones y semejantes las interacciones entre los mismos.

Para medir la performance a lo largo del tiempo proponemos dos medidas. El *ratio de simulación total* ($sr_T(t)$) definido como el promedio entre el tiempo de simulación (t) y el tiempo real transcurrido ($RT(t)$)

$$sr_T(t) = t/RT(t) \tag{5.1}$$

y el *ratio de simulación parcial* ($sr_P(t)$) definido como un promedio móvil entre el tiempo de simulación y el tiempo real de los últimos segundos de simulación

$$sr_P(t) = (t-\gamma)/(RT(t) - RT(t-\gamma)) \tag{5.2}$$

donde γ es un intervalo de tiempo fijo.

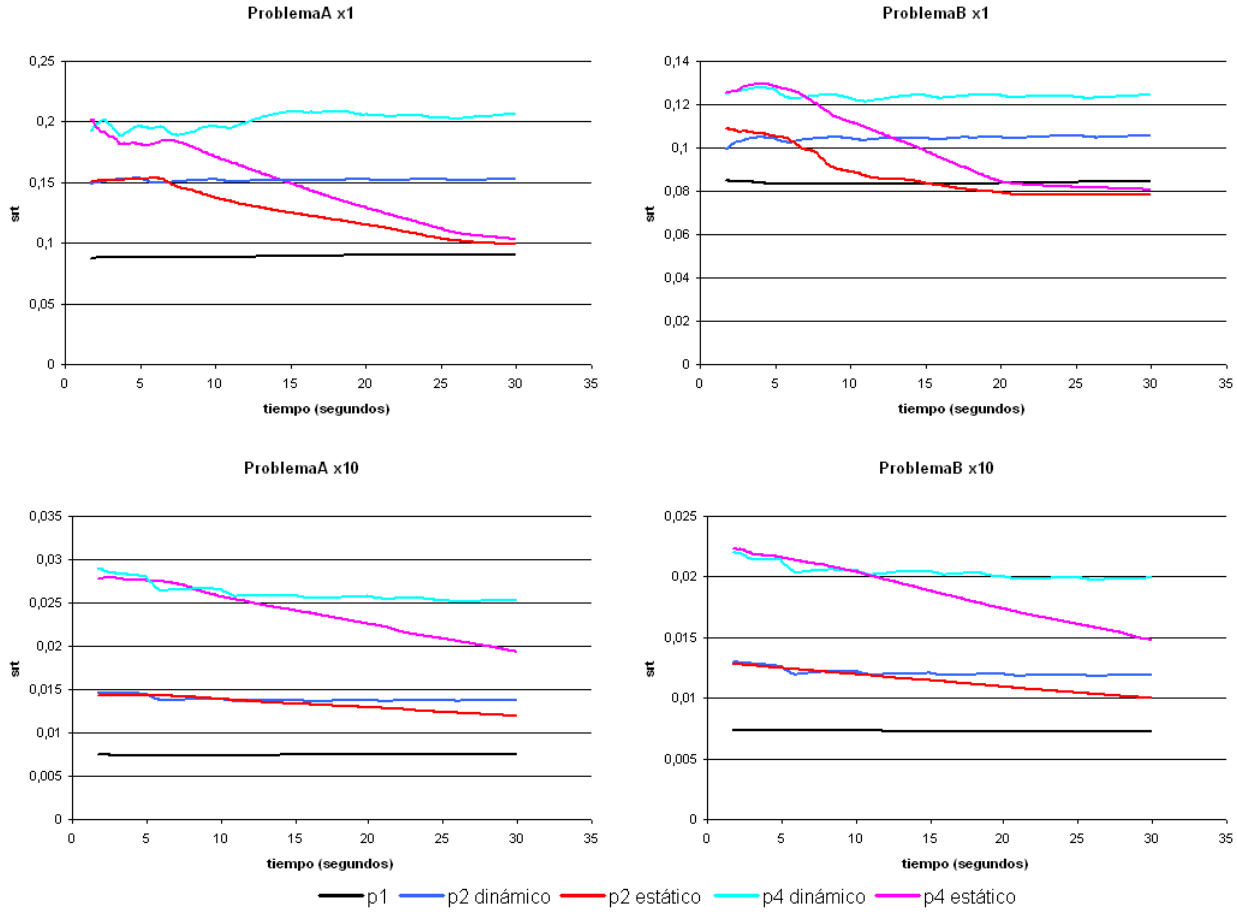


Figura 5.5: $sr_T(t)$ para los problemas A y B $x1$ y $x10$ para $p1$, $p2$, $p4$ utilizando balanceo estático y dinámico de la carga.

En la figura 5.5 se muestra la evolución del sr_T en el tiempo para los problemas A y B de tamaño $x1$ y $x10$ para $p1$, $p2$ y $p4$ utilizando balance de carga dinámico y estático. Para ambos problemas $x1$ y $x10$, se ve que el sr_T para un único procesador se mantiene constante a lo largo del tiempo. Esto se debe a que la cantidad de cálculos correspondientes a las fuerzas de interacción se mantienen constantes, permitiendo comparar las simulaciones para más procesadores utilizando balanceo estático y dinámico únicamente por cálculos y comunicaciones realizados para mantener el sistema en forma distribuida. Comparando la evolución del sr_T utilizando balanceo estático y dinámico, se ve que a partir de cierto instante de tiempo, el dinámico comienza a ser mejor. Para los problemas más chicos esta diferencia se hace mayor pudiéndose deber a que el nivel de desbalanceo también es mayor. En la figura 5.6 se muestra el estado final del Problema A $x10$ donde se ve que el nivel de desbalanceo no es muy grande, comparado con

el último gráfico de la figura 5.2, donde se encuentra desbalanceado completamente al contar con todos los peatones asignados a un único procesador. Por este mismo motivo, en los problemas $x1$ el ratio decae al ratio observado en las simulación utilizando un único procesador, mientras que en los problemas $x10$ no. Si continuáramos la simulación por más tiempo, el ratio utilizando balanceo estático seguiría decayendo hasta alcanzar al de un procesador, mientras que utilizando balanceo dinámico mantendría una tendencia constante.

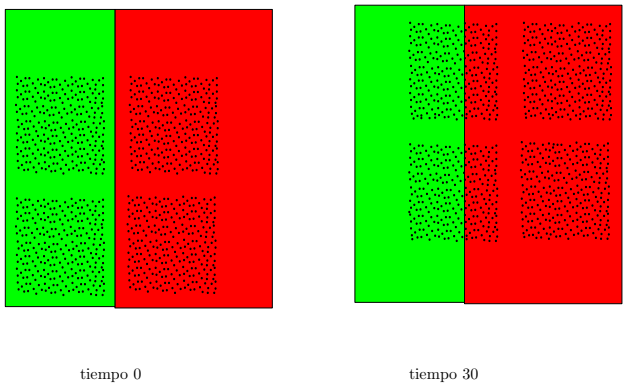


Figura 5.6: Estado final del Problema A $x10$, se ve que el porcentaje de desbalanceo no es muy grande, comparado con el último gráfico de la figura 5.2 donde se encuentra desbalanceado completamente.

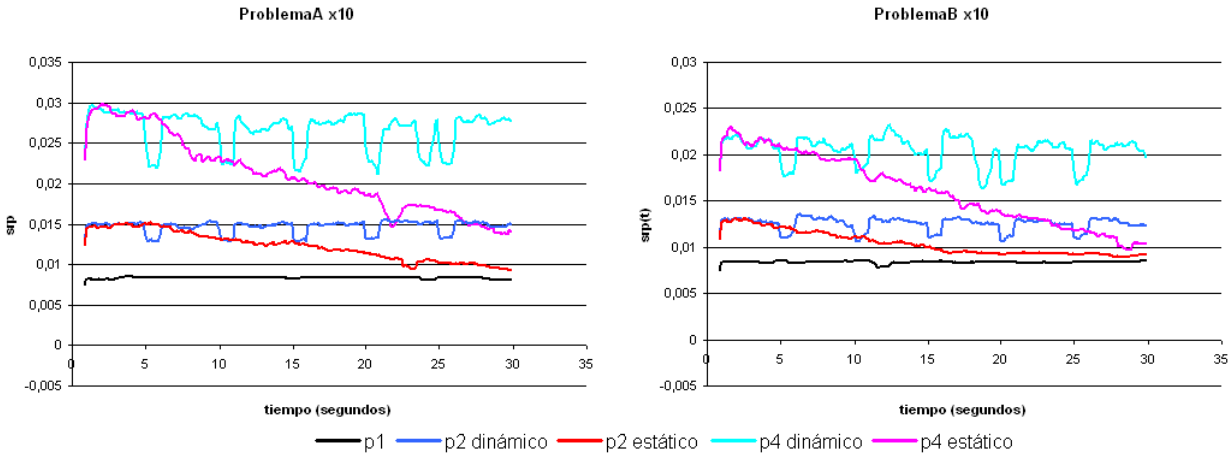


Figura 5.7: $sr_P(t)$ para los problemas A y B $x10$ para $p1$, $p2$ y $p4$ utilizando balanceo estático y dinámico de la carga.

Analizar el sr_P permite ver la performance de la simulación a lo largo del tiempo, teniendo en cuenta únicamente los últimos segundos. Mientras que el sr_T brinda una

visión global de la performance de la simulación, el sr_P brinda una visión más instantánea. En la figura 5.7 se muestra el sr_P en función del tiempo para los problemas A y B $x10$ para $p1$, $p2$ y $p4$, también utilizando balanceo estático y dinámico. Se ve como el sr_P decae en el momento de reparticionar, pero luego vuelve a los valores anteriores. Dependiendo del tamaño del problema, varía el costo del algoritmo de repartición; haciendo que estos decaimientos en el sr_P sean dependientes de la cantidad de peatones. En esta figura también se ve como el sr_P para balanceo estático decae, tendiendo al ratio de la simulación con un único procesador. En este caso se ve que se acerca más a este valor que en el caso del ratio total (tener en cuenta que se trata del problema $x10$, ver figura 5.5), ya que la buena performance del principio no influye en las mediciones del ratio parcial.

Hasta acá comparamos la performance de la simulación a lo largo del tiempo para las variantes de balanceo estático y dinámico. A continuación compararemos los distintos problemas con distintos tamaños y cantidades de procesadores en función del tiempo total que insumió la simulación. Como en los casos anteriores se obtuvieron mejores resultados utilizando balanceo dinámico de la carga, en las siguientes comparaciones utilizaremos únicamente esta variante para cada problema.

En el cuadro 5.1 se muestran los tiempos de simulación obtenidos para los distintos problemas. En la figura 5.8 se muestra el tiempo de simulación en función del tamaño del problema (a) y en función de la cantidad de procesadores (b). En la figura (a), se ve que el tiempo crece a medida que se agranda el tamaño del problema. También se ve que para un procesador, en ambos problemas, el tiempo de simulación es semejante, mientras que para más procesadores la diferencia es mayor, observando que para el problema A se registra un tiempo menor. Esto último puede deberse a que por la separación en bloques que fue planteada en el problema A y la partición obtenida (ver figura 5.3), se realizan menos comunicaciones entre los procesadores. En la figura (b) se puede observar como disminuye el tiempo cuando se agregan procesadores. No siendo el caso del problema $x1$, ya que la cantidad de cálculos realizados no justifica el uso de varios procesadores. También se ve claramente como la diferencia de tiempos entre $p1$ y $p2$ es mayor que entre $p2$ y $p4$.

Para un mayor análisis de la performance obtenida en una simulación distribuida con respecto a su versión centralizada utilizamos tres medidas de uso frecuente: *aceleración* (speedup), *eficiencia* (efficiency) y *sobrecarga* (overhead). Definimos la aceleración como cuánto más rápido es la simulación distribuida del problema de tamaño x con p procesadores con respecto a la simulación del mismo problema con un procesador,

	Tiempo (minutos)					
	Problema A			Problema B		
	x1	x5	x10	x1	x5	x10
p1	5.53	31.92	66.35	5.9	33.09	68.76
p2	3.26	17.31	36.27	4.74	20.92	41.96
p4	2.41	9.45	19.72	4.01	13.11	25.14

Cuadro 5.1: Tiempos medido en minutos.

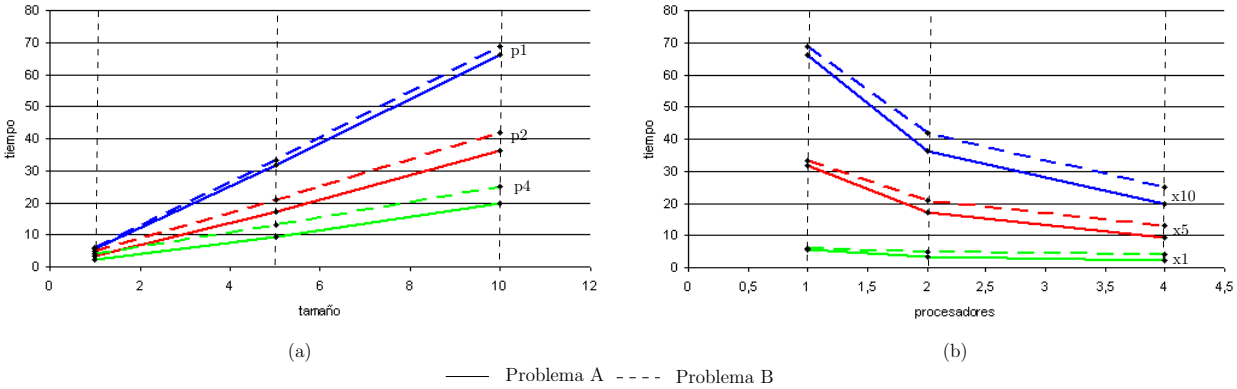


Figura 5.8: (a) Tiempo en función del tamaño del problema. (b) Tiempo en función de la cantidad de procesadores.

dada por la siguiente expresión

$$S(p,x) = T(1,x)/T(p,x) \tag{5.3}$$

donde $T(p,x)$ es el tiempo que tarda una simulación del problema de tamaño x con p procesadores. A una relación de $S(p,x) = p$ se la denomina *aceleración lineal*, la cual representa la aceleración ideal que puede obtenerse al descomponer el problema en forma distribuida. Definimos eficiencia como la utilidad de cada procesador durante una simulación, dada por la siguiente expresión

$$E(p,x) = S(p,x)/p \tag{5.4}$$

Una eficiencia igual a 1 representa la utilidad máxima y una de valor 0 la mínima. Definimos sobrecarga como la cantidad de trabajo realizado por la simulación distribuida que no es realizada por la versión centralizada, dada por la siguiente expresión

$$O(p,x) = T(p,x).p - T(1,x) \tag{5.5}$$

Este trabajo adicional puede deberse a distintos motivos, como ser: la comunicación entre los procesadores, los cálculos extras al distribuir el problema (incluidos los necesarios para lograr la distribución) y los tiempos ociosos de los procesadores.

En el cuadro 5.2 se muestran los valores obtenidos para las distintas simulaciones realizadas para las tres medidas de performance analizadas. En la figura 5.9 se muestra, en tres gráficos distintos, la relación entre los valores obtenidos y la cantidad de procesadores utilizados para cada simulación. Se puede ver como la aceleración aumenta al agregar más procesadores pero la eficiencia decae. En la figura 5.8(b) habíamos visto que las pendientes eran distintas al pasar de $p1$ a $p2$ que al pasar de $p2$ a $p4$, esto se ve reflejado en la figura 5.9(b) donde la eficiencia para $p2$ es mayor que para $p4$.

Para el problema A $x5$ y $x10$ la aceleración -y la eficiencia- obtenida es la misma, ya sea para $p2$ o $p4$, y superior que para el problema B. Esto se debe a que por la separación en bloques que fue planteada para este problema, se obtiene una menor cantidad de comunicaciones, y por ende, una sobrecarga menor. Para el problema B $x1$ la aceleración no aumenta mucho al agregar más procesadores. Esto se puede deber a que la sobrecarga que se agrega al tener más procesadores significan un porcentaje del tiempo total muy grande frente a la poca cantidad de cálculos que se realizan. Para el problema A $x1$, la aceleración es un poco mejor, ya que tiene menos comunicaciones, aunque muy lejana a la ideal.

	Aceleracion						Eficiencia						Sobrecarga					
	Problema A			Problema B			Problema A			Problema B			Problema A			Problema B		
	x1	x5	x10	x1	x5	x10	x1	x5	x10	x1	x5	x10	x1	x5	x10	x1	x5	x10
p2	1.69	1.84	1.82	1.24	1.58	1.63	0.84	0.92	0.91	0.62	0.79	0.81	0.98	2.71	6.19	3.58	8.74	15.15
p4	2.29	3.37	3.36	1.47	2.52	2.73	0.57	0.84	0.84	0.36	0.63	0.68	4.11	5.88	12.53	10.14	19.36	31.8

Cuadro 5.2: Valores de aceleración, eficiencia y sobrecarga para las distintas simulaciones realizadas.

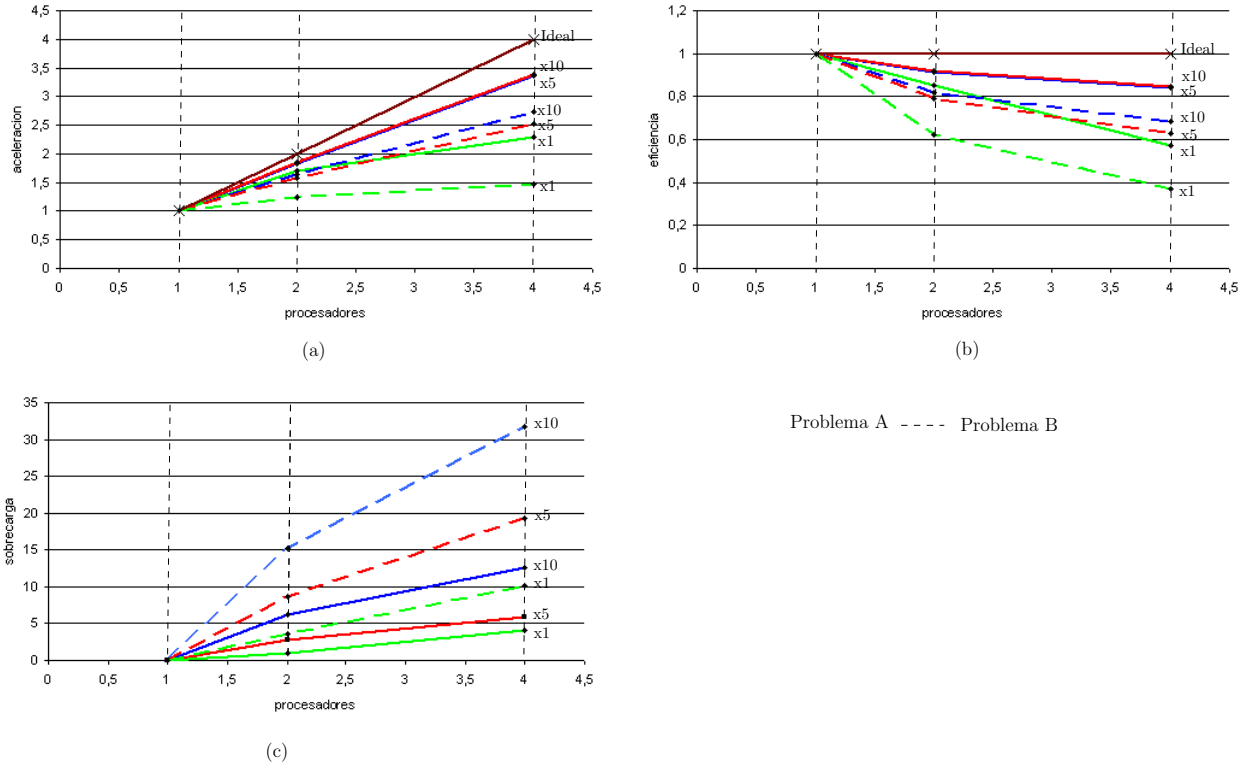


Figura 5.9: (a) Aceleración en función de la cantidad de procesadores (b) Eficiencia en función de la cantidad de procesadores. (c) Sobrecarga en función de la cantidad de procesadores.

Capítulo 6

Conclusiones y Trabajo Futuro

Primero es importante remarcar que nuestro objetivo principal de simulación distribuida con agentes utilizando balanceo dinámico de carga fue resuelto satisfactoriamente. La utilización del paradigma de agentes para la simulación de sistemas autoorganizados permite mantener un modelo que refleje la realidad. Modelar las partes del sistema como agentes hace que aparezcan diversas unidades de cálculo relacionadas entre sí dependiendo de las relaciones existentes en el sistema real. La simulación de sistemas compuestos por miles de agentes requiere una gran cantidad de recursos. Como se vió en este trabajo, la utilización de un cluster de computadoras aparece como una potente herramienta para llevar a cabo este tipo de simulaciones. El problema a resolver, para un mejor aprovechamiento del cluster, es la correcta distribución de los agentes y la comunicación entre quienes se encuentren asignados a distintos recursos.

En este trabajo se analizó y desarrolló un modelo computacional de tráfico peatonal implementado sobre un framework distribuido de simulación basada en agentes. Si bien se planteó un modelo particular para la simulación de tráfico peatonal, los conceptos aquí presentados pueden ser aplicados para la distribución de otros sistemas, mediante una adecuada elección del método de creación del grafo de interacciones.

Para que un sistema distribuido sea más eficiente que uno centralizado, la sobrecarga debe ser suficientemente pequeña. En este trabajo se introdujeron distintos mecanismos para reducirla. Para disminuir la sobrecarga por comunicación se agruparon todos los mensajes entre agentes de distintos nodos, existiendo un único mensaje entre pares de nodos vecinos para cada instante de tiempo. De esta forma, se reducen las comunicaciones, teniendo un orden mucho menor que $O(p^2)$, siendo p la cantidad de procesadores, y no dependientes de la cantidad de agentes a comunicar. Para reducir la sobrecarga por

los cálculos extra, se atacaron los distintos problemas que surgieron al tener los datos distribuidos para ser resueltos eficientemente y que agreguen el menor costo posible. Al tener una distribución por regiones, muchos de los cálculos extra corresponden a problemas geométricos, los cuales requieren una gran cantidad de cálculos. Por otro lado, aparece el costo de calcular la partición del espacio para balancear la carga el cual no existía en la versión centralizada. Finalmente, la sobrecarga por los tiempos ociosos fue disminuida realizando diversos mecanismos de paralelización y sincronización para aprovechar los tiempos de comunicación, para realizar cálculos con los datos locales y fundamentalmente, aplicando técnicas de balanceo dinámico, para equiparar la carga entre los distintos procesadores.

A partir de los resultados obtenidos, se ve como al agregar más procesadores aumenta la aceleración pero disminuye la eficiencia. Debido a los costos de la sobrecarga y dependiendo de las características del problema, no siempre agregar procesadores implicará una reducción de los tiempos. Para cada situación será necesario evaluar los costos de agregar nuevos recursos frente a las ventajas de obtener los resultados en un menor tiempo.

Como conclusión de este trabajo, se pudo observar que la simulación distribuida permite resolver problemas de gran envergadura y en menor tiempo. A partir de los resultados, se aprecia claramente como la introducción del balanceo dinámico juega un rol fundamental en la distribución de problemas altamente dinámicos.

En este trabajo el usuario tiene que determinar cada cuánto tiempo desea rebalancear la carga. Queda como trabajo futuro la introducción de mecanismos que permitan la automatización de este proceso. Una posibilidad puede ser monitorear la performance de cada procesador y rebalancear al detectar alguna desviación del promedio.

Los algoritmos utilizados balancean la carga en forma equitativa para todos los procesadores, suponiendo que estamos trabajando con un cluster homogéneo. Esto no siempre es así ya que muchos de los clusters están formados por procesadores con distintas velocidades y memorias, haciendo que los procesadores más rápidos tengan tiempos ociosos. Para solucionar este problema se podría modificar el algoritmo de partición del espacio para adaptarse a la heterogeneidad de procesadores. Esto no pareciera presentar mayores dificultades, ya que los algoritmos de partición de grafos vistos presentan esta alternativa para generar particiones de distintos tamaños, indicándole las diferencias de velocidades entre los procesadores en el momento de ser invocados.

Dado el fuerte impacto de las regiones resultantes del algoritmo de particionamiento en el balance de carga, queda como trabajo futuro probar con distintas formas de

creación de regiones. Por ejemplo, variar las formas de creación del grafo de interacciones y funciones de asignación de pesos a los ejes, pudiéndose adaptar a las distintas distribuciones de peatones en el espacio. En este trabajo utilizamos una función que decae a medida que la distancia entre los agentes aumenta, pero, como decaería debe depender de la densidades resultantes en cada problema y no ser ingresadas por el usuario. También se pueden utilizar distintas funciones para la asignación de pesos a los vértices, que representa la cantidad de cómputos que demanda cada unidad. De esta forma se pueden representar mejor los problemas donde no todos los peatones tienen la misma cantidad de interacciones.

Apéndice A

Partición de Grafos

Los *algoritmos de partición de grafos* son utilizados en los sistemas distribuidos que cuentan con muchas unidades de cálculo. Para lograr un balance de la carga, el problema se modela con un grafo donde los vértices representan las unidades de cálculo y los ejes las interdependendencias entre los mismos. La partición del grafo se utiliza para asignar las distintas unidades de cálculo a los distintos procesadores del cluster de computadoras. Para lograr el objetivo de distribuir la carga del sistema, se debe lograr una asignación de unidades de cálculo en procesadores que permita obtener la mejor performance posible. El algoritmo de partición debe lograr asignar aproximadamente la misma cantidad de trabajo para cada procesador y minimizar las comunicaciones necesarias por la interdependencia entre los procesos. Esto se alcanza calculando una partición tal que asigne la misma cantidad de vértices a cada subdominio y minimice la cantidad de ejes que unen vértices de distintos subdominios. Ejemplos donde se utiliza partición de grafos son la solución numérica de ecuaciones diferenciales parciales utilizando el método de los elementos finitos, solución de sistemas lineales esparsos vía métodos iterativos, entre otros [4, 5, 6].

El problema de partición es NP-Completo, pero sin embargo existen distintas heurísticas que obtienen particiones de una buena calidad para ser utilizadas en sistemas distribuidos. Aún la bisección, partición en dos subdominios, es un problema que sigue siendo NP-Completo. Muchos algoritmos utilizan la bisección como base. Para obtener una partición en p partes realizan bisecciones en forma recursiva. Para particionar en una cantidad de subdominios que no sea potencia de dos, la bisección calcula subdominios de distintos tamaños.

Obtener una única partición del grafo y descomponer el problema no es suficiente

para hacer un buen uso del cluster. Los cálculos realizados pueden ir evolucionando con el transcurso del tiempo y la interdependencia de los mismos cambiar, por lo que se obtendría un incremento en las comunicaciones o un desbalance de la carga. Para solucionar este nuevo problema se introduce lo que se conoce como *balanceo dinámico de la carga*, resuelto utilizando *repartición de grafos*. A los objetivos de distribuir la carga aproximadamente igual para cada subdominio minimizando las comunicaciones, se le agrega el objetivo de minimizar las diferencias con la partición anterior.

Hay varias cuestiones a tener en cuenta en las técnicas de repartición [7]. La primera de ellas es el costo computacional que demanda calcular la nueva partición. Como este algoritmo va a ser ejecutado periódicamente, con el objetivo de mantener balanceada la carga, su costo computacional debe ser bajo, o por lo menos significativamente menor que el tiempo que demandan los cálculos del problema que se está resolviendo. La segunda, la repartición debe minimizar la cantidad de elementos que son necesarios migrar de un procesador a otro luego de obtener una nueva partición. La tercera y última es el paralelismo, ya que los datos se encuentran distribuidos en los procesadores, es conveniente que calcule la partición sin la necesidad de tener que migrar todos los datos a un procesador central. A los algoritmos de repartición que toman en consideración los puntos mencionados en el párrafo anterior se los denominan como casi dinámicos o sincrónicos, porque calculan la partición en forma colectiva. Una forma más dinámica, o asincrónica es que el algoritmo que resuelve el problema balancee la carga a medida que sea necesario. Luego de cada iteración de cálculo podría migrar la carga excedente a los otros procesadores vecinos.

La partición de un grafo se puede obtener con distintas técnicas, en [3, 5] se pueden encontrar varios de estos métodos. Los algoritmos geométricos son los que utilizan las posiciones de los vértices para calcular la partición. Estos son los más rápidos, pero no siempre se pueden utilizar. Por otro lado, no tienen en cuenta los ejes del grafo, por lo que no se minimiza la cantidad de ejes que cruzan de subdominio y no siempre generan particiones de buena calidad. Por otro lado, están los algoritmos multinivel que consisten en reducir el grafo para luego aplicar el algoritmo de partición en un grafo más chico y luego volver al grafo original aplicando técnicas de refinamiento que mejoran la calidad de la partición (ver figura A.1). Para un mayor detalle de estas técnicas ver [4].

Para obtener una repartición del grafo se utilizan dos esquemas distintos: *partición desde cero* y *partición difusiva*. Ambos esquemas pueden verse también en [3, 5] y son comparados en [10]. La primera plantea utilizar los mismos métodos de partición

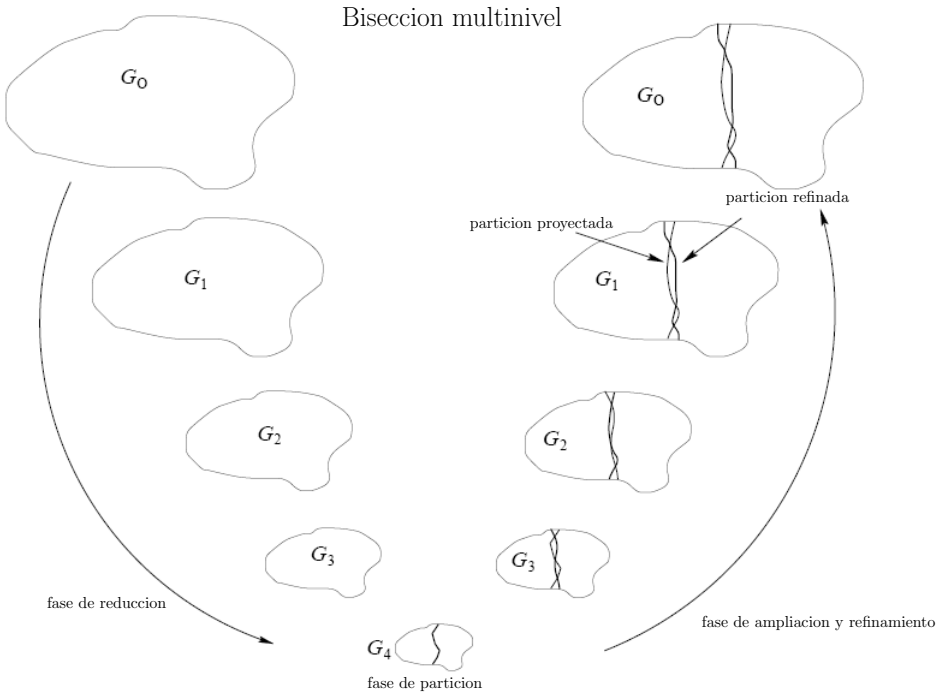


Figura A.1: Esquema de un método multinivel divido en tres fases: (a) reducción, (b) partición y (c) ampliación y refinamiento.

estática y luego asignar inteligentemente las etiquetas de los subdominios de forma que se minimicen las migraciones (ver figura A.2). La segunda, se basa en hacer cambios incrementales en la partición original hasta llegar a una partición balanceada (ver figura A.3). En [8, 9] se introducen varios conceptos y cuestiones a tener en cuenta en los algoritmos difusivos.

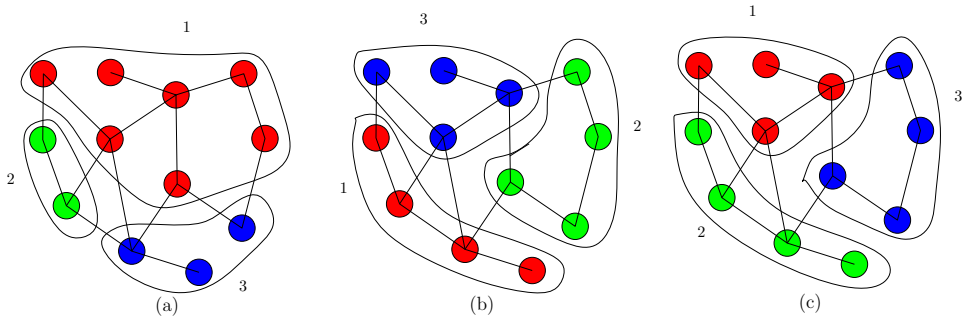


Figura A.2: Ejemplo de una partición desde cero. (a) partición original, (b) nueva partición y (c) partición con las etiquetas reasignadas.

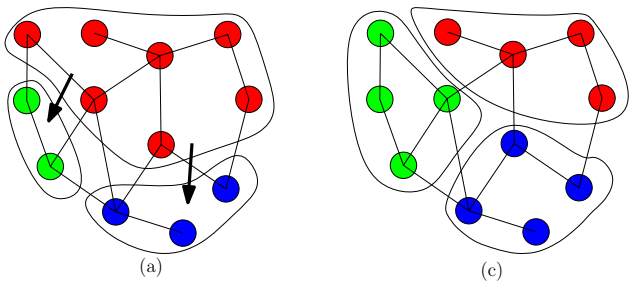


Figura A.3: Ejemplo de una partición difusiva. (a) partición original y (b) partición obtenida luego de la difusión del subdominio mas sobrecargado.

Apéndice B

Glosario

ri (Radio de Influencia) Distancia a partir de la cual las fuerzas son despreciables.

$F(p)$ Fuerza que influye sobre el peatón p.

$F_V(p)$ Fuerza restringuida a un conjunto de peatones que influye sobre el peatón p.

$V_d^t(p)$ Conjunto de peatones y obstáculos que se encuentran a una distancia menor que d del peatón p para el instante de tiempo t. Se los llama vecinos a distancia d de p.

$FD(p)$ Fuerza del Deseo del peatón p.

$FS_V(p)$ Fuerza social del peatón p con los vecinos V.

$FG_V(p)$ Fuerza granular del peatón p con los vecinos V.

K Cada cuántas iteraciones se descubren vecinos.

Δt Paso de discretización del tiempo.

$vmax$ Velocidad máxima de los peatones.

δ Distancia utilizada para calcular la vecindad de un peatón para un rango de tiempo. $\delta = vmax * K * \Delta t * 2 + ri$ para el rango de tiempo $[t, t + k * \Delta t)$.

PI (Probabilidad de Interacción) probabilidad de que dos agentes se envíen mensajes.

$V_{\delta,N}^t(p)$ Conjunto de peatones vecinos asignados al nodo N. $V_{\delta(p),N}^t = \{\ v : \ \textit{peatón} : v \in V_\delta^t(p) \wedge v \in N\ \}$

N_p Nodo al que está asignado el peatón p.

$\Psi(N)$ Conjunto de peatones asignados al nodo N.

$V_\delta^t(N)$ Conjunto de vecinos de los peatones asignados al nodo N. $V_\delta^t(N) = \bigcup_{p \in N} V_\delta^t(p)$.

$\mathcal{N}_\delta^t(N, M)$ Conjunto de peatones que el nodo N debe notificar al nodo M en el instante t. $\mathcal{N}_\delta^t(N, M) = \Psi(N) \cap V_\delta^t(M)$.

$\mathbb{N}(N, M)$ Región de notificación, contiene a los peatones $\mathcal{N}_\delta^t(N, M)$.

$\mathcal{M}^t(N, M)$ Conjunto de peatones que el nodo N debe migrar al nodo M en el instante t.

$\mathbb{M}(N, M)$ Región de migración, contiene a los peatones $\mathcal{M}^t(N, M)$.

$Bfr_d(A)$ Ampliación de la región A en una distancia d. $Bfr_d(A) = \{\ x \in \mathbb{R}^2 : \exists y \in A : ||x - y|| \leq d\ \}$.

$region(N)$ Región asignada al nodo N.

τ Parámetro utilizado para calcular el tamaño de la región de migración, $\tau = K * \Delta t * vmax$.

$V(N)$ Conjunto de nodos vecinos del nodo N. $V(N) = \{\ M : \mathbb{N}(N, M) \neq \emptyset \vee \mathbb{M}(N, M) \neq \emptyset\ \}$

bounds Rectángulo más pequeño que contiene totalmente a una región.

Bibliografía

[1] Haile, J. M., *Molecular dynamics simulation: elementary methods*, 1. ed, Wiley, 1992

[2] Frenkel, D.; Smit, B., *Understanding Molecular Simulation from Algorithms to Applications*, Computational Science Series, 2001

[3] Schloegel, K; Karypis, G. and Vipin K., *Graph partitioning for high performance scientific simulations*, In J. Dongarra et al., editor, CRPC Parallel Computing Handbook. Morgan Kaufmann, 2000.

[4] Karypis, G. and Kumar, V., *A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs*, SIAM Journal on Scientific Computing, 1998

[5] Hu, Y. F. and Blake, R. J., *Load balancing for unstructured mesh applications*, In Progress in Computer Research, F. Columbus, Ed. Nova Science Publishers, 2001

[6] Castaños, J. G. and Savage, J. G., *Repartitioning Unstructured Adaptive Meshes*, In Proc. Intl. Parallel and Distributed Processing Symposium, 2000

[7] Walshaw, C.; Cross, M. and Everett, M. G., *Dynamic mesh partitioning: A unified optimisation and load-balancing algorithm*, Technical Report 95/IM/06, Centre for Numerical Modelling and Process Analysis, University of Greenwich, 1995.

[8] Hu, Y. F., Blake, R. J. and Emerson, D. R., *An optimal migration algorithm for dynamic load balancing*, Concurrency: Prac. and Exper. 10, 1998

[9] Hu, Y. F. and Blake, R. J., *An improved diffusion algorithm for dynamic load balancing*, Parallel Comput., Elsevier Science Publishers B. V., 1999

[10] Schloegely, K.; Karypisz, G.; Kumarx, V.; Biswas, R. and Olikerk, L., *A Performance Study of Diffusive vs. Remapped Load-Balancing Schemes*, ISCA 11th Int'l Conference on Parallel and Distributed Computing Systems, 1998.

[11] Karypis, G. and Kumar, V., *METIS. A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices*, http://glaros.dtc.umn.edu/gkhome/metis/metis/download, 1998

[12] Helbing, D.; Farkas, I. and Vicsek T., *Simulating dynamical features of escape panic*, Nature, 2000

[13] Gloor, C.; Mauron, L. and Kai N., *A pedestrian simulation for hiking in the alps*, 3rd Swiss Transport Research Conference, 2003

[14] Rajkumar B., *High Performance Cluster Computing: Architectures and Systems, Volume 1*, 1st edition, Prentice Hall PTR, 1999

[15] Pacheco, P. S., *Parallel Programming With MPI*, Morgan Kaufamnn Publishers, Inc., 1997

[16] Samet, H., *The quadtree and related hierarchical data structures*, ACM Computer Surveys 16(2):187-260, 1984

[17] Samet, H., *Spatial Data Structures*, Reading, Addison-Wesley/ACM, 1995

[18] Roussopoulos, N.; Kelley S. and Vincent, F., *Nearest Neighbor Queries*, Proceedings of ACM Sigmod , 1995

[19] Papadopoulos, A. and Manolopoulos, Y., *Performance of Nearest Neighbor Queries in R-Trees*, Proceedings of the 6th International Conference on Database Theory, 1997

[20] Jonson, S., *Sistemas Emergentes, Turner*, Fondo de Cultura Económica, 2001.

[21] Heylighen, F., *Principia Cibernética Web*, http://pespmc1.vub.ac.be/SELFORG.html; 1997.

[22] Heylighen, F., *Self-organization, Emergence and the Architecture of Complexity*, Transdisciplinary Research Group, Free University of Brussels.

[23] Arkin R. C., *Behavior-Based Robotics*, MIT Press, 1998.

[24] Wooldrige M., *Agent-based software engineering*, IEEE Proc. On Software Engineering, 144 (1) 26-37; 1997.

[25] Gilman M., *Simulación Basada en Agentes y sus Beneficios*, Reporte Interno Urbix Technologies, 2005.

[26] Mas A., *Agentes Software y Sistemas Multiagente. Conceptos, Arquitecturas y Aplicaciones*, Pearson Educacion, 2005

[27] Aurenhammer F. and Klein R., *Voronoi Diagrams*, Handbook on Computational Geometry, Elsevier, to appear

[28] Fruin, J. J., *Pedestrian Planning and Design*, Elevator World, New York, 1971

[29] Hankin B. D. and Wright R. A., *Passenger flow in subways*, Operational Research Quarterly, 1958