Búsqueda de errores en programas usando Java PathFinder y ACOhg

LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

Francisco Chicano y Enrique Alba

Motivación

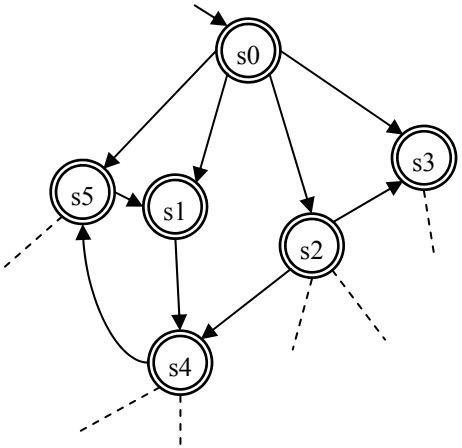
- El Software actual es difícil de testar por simple inspección ...
- ... y se encuentra en el núcleo de gran cantidad de sistemas críticos

- Hay que acudir a técnicas para demostrar la corrección del software concurrente
- Model checking → completamente automático
- Las técnicas existentes tienen problemas con los grandes programas
- En este trabajo presentamos ACOhg para buscar errores usando JPF

Model checking explícito

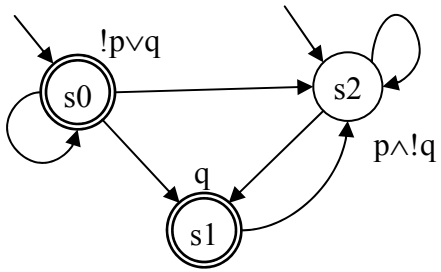
- **Objetivo: demostrar que un programa M satisface una propiedad f : $M \models f$**
- **En el caso más general *f* es una fórmula en lógica temporal (LTL, CTL, etc.)**

Programa M



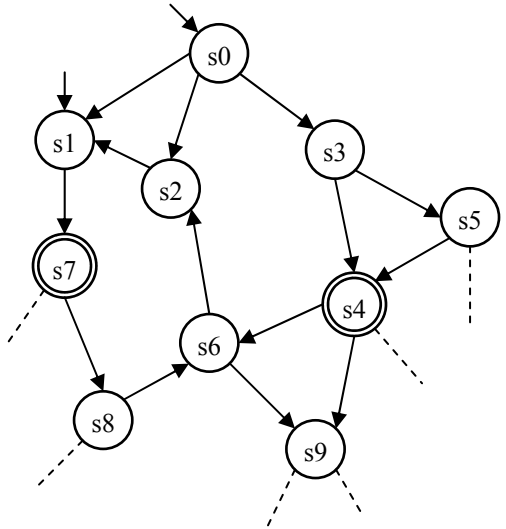
$\cap$

Fórmula LTL $\neg f$ (*never claim*)



$=$

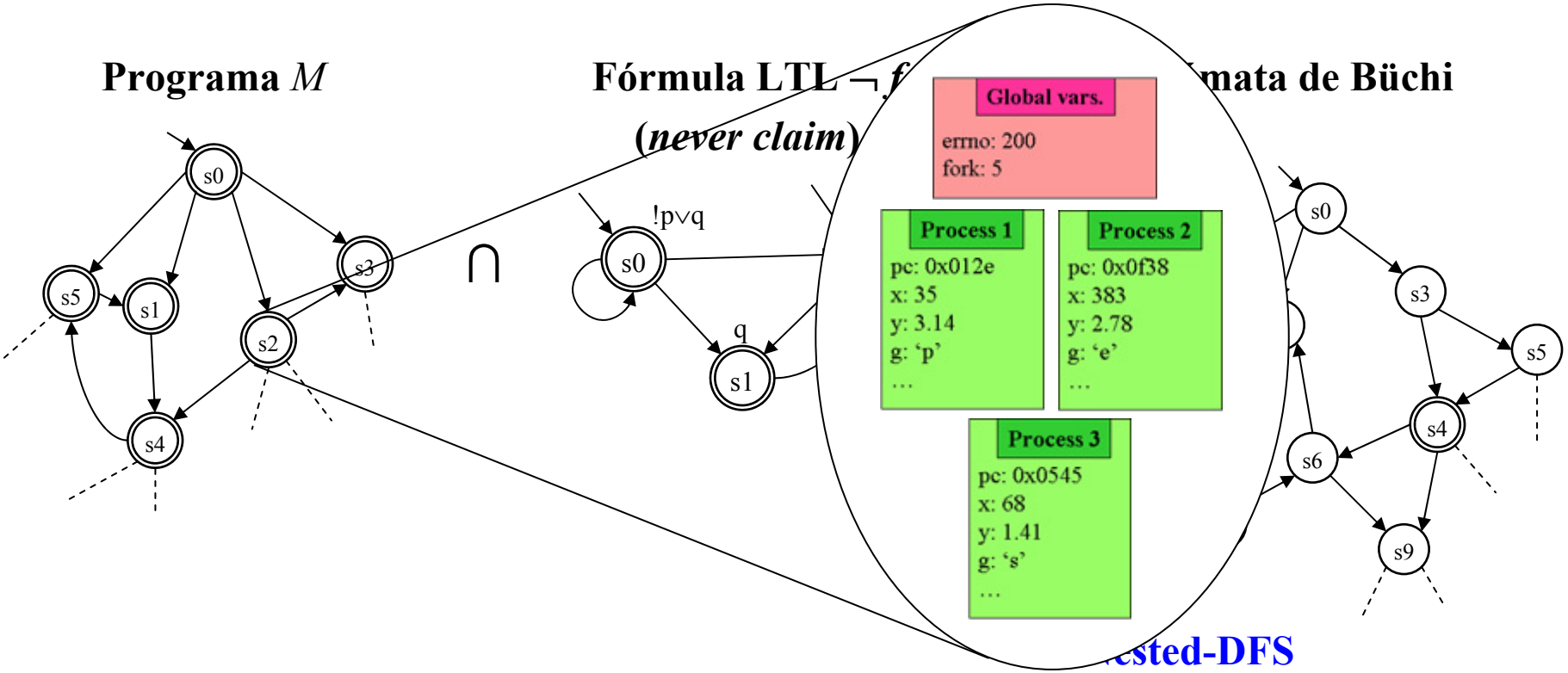
Autómata de Büchi



Nested-DFS

Model checking explícito

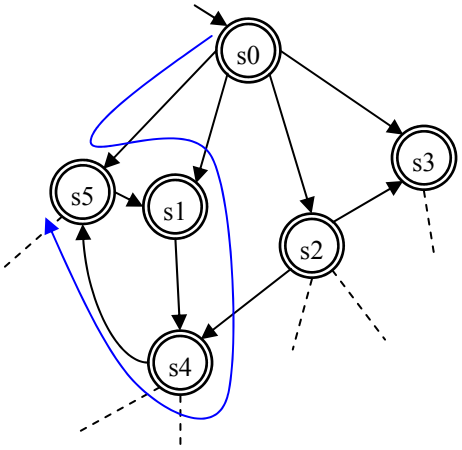
- **Objetivo: demostrar que un programa M satisface una propiedad f : $M \models f$**
- **En el caso más general f es una fórmula en lógica temporal (LTL, CTL, etc.)**



Model checking explícito

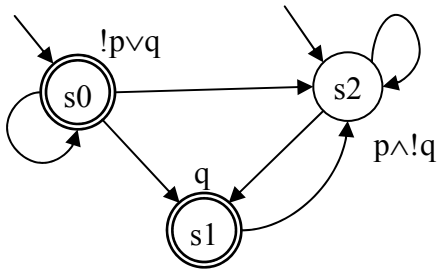
- **Objetivo: demostrar que un programa M satisface una propiedad f : $M \models f$**
- **En el caso más general *f* es una fórmula en lógica temporal (LTL, CTL, etc.)**

Programa M



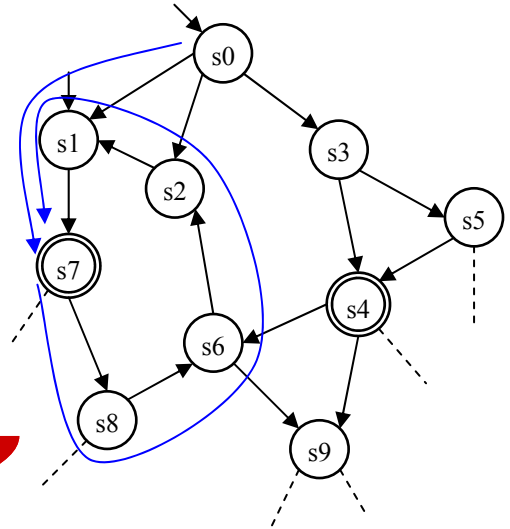
$\cap$

Fórmula LTL $\neg f$ (*never claim*)



$=$

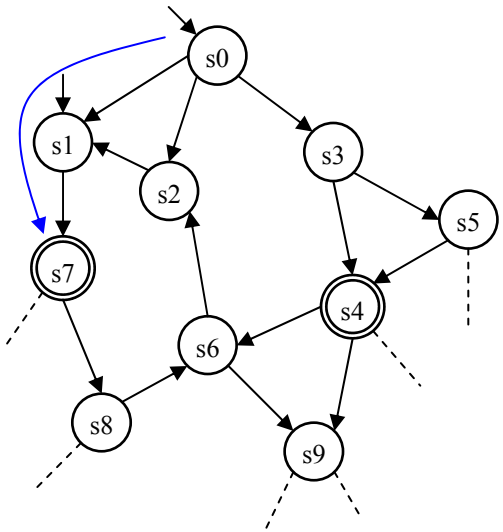
Autómata de Büchi



Nested-DFS

Propiedades de seguridad

$$\forall \sigma \in S^{\omega} : \sigma \nvdash \mathcal{P} \Rightarrow (\exists i \geq 0 : \forall \beta \in S^{\omega} : \sigma_i \beta \nvdash \mathcal{P})$$



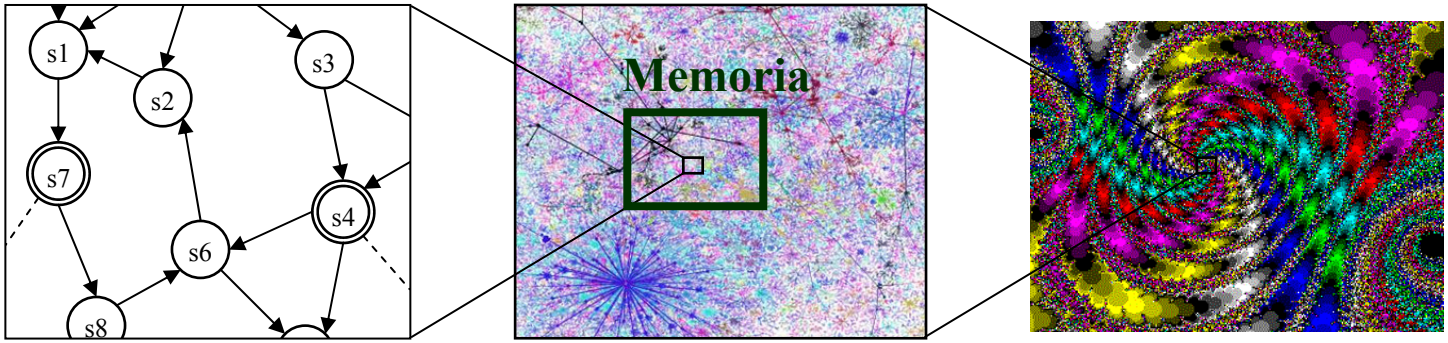
Propiedades en JPF

- Excepciones
- Interbloqueos

- **Una traza de error es un camino hasta un estado de error**
- **El problema se convierte en una búsqueda de un nodo en un grafo (DFS, BFS)**

Problema de la explosión de estados

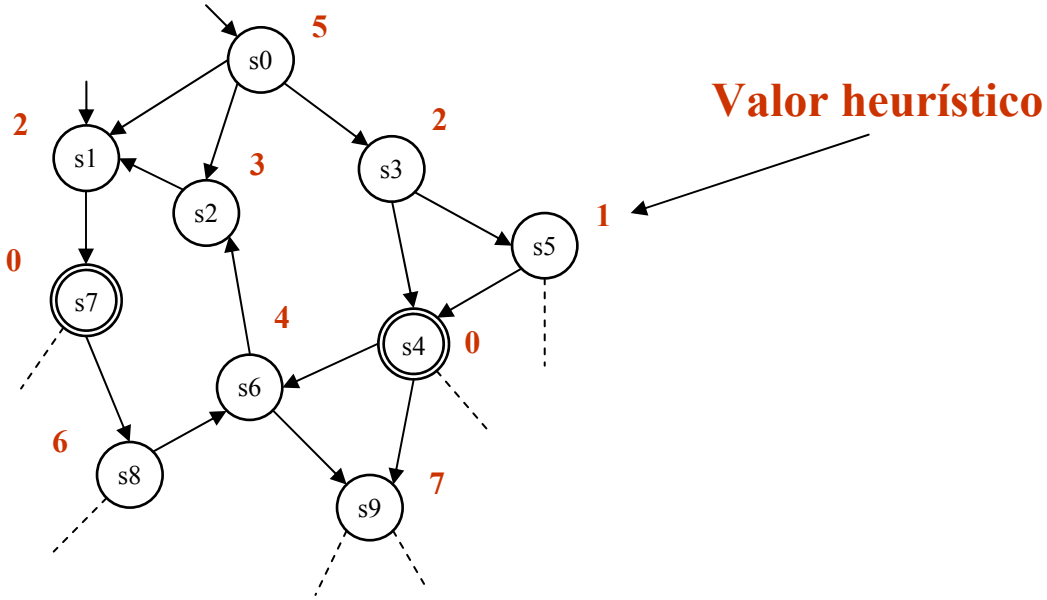
- El número de estados es muy grande incluso en pequeños programas



- Ejemplo: modelo del problema de los filósofos con *n* filósofos → 3^n estados
- Por cada estado hay que almacenar el heap y las pilas de las distintas hebras
- Soluciones: compresión por colapso, representación mínima de autómatas, bitstate hashing, reducción de orden parcial, reducción de simetría
- Los grandes programas no se pueden verificar pero se pueden encontrar errores

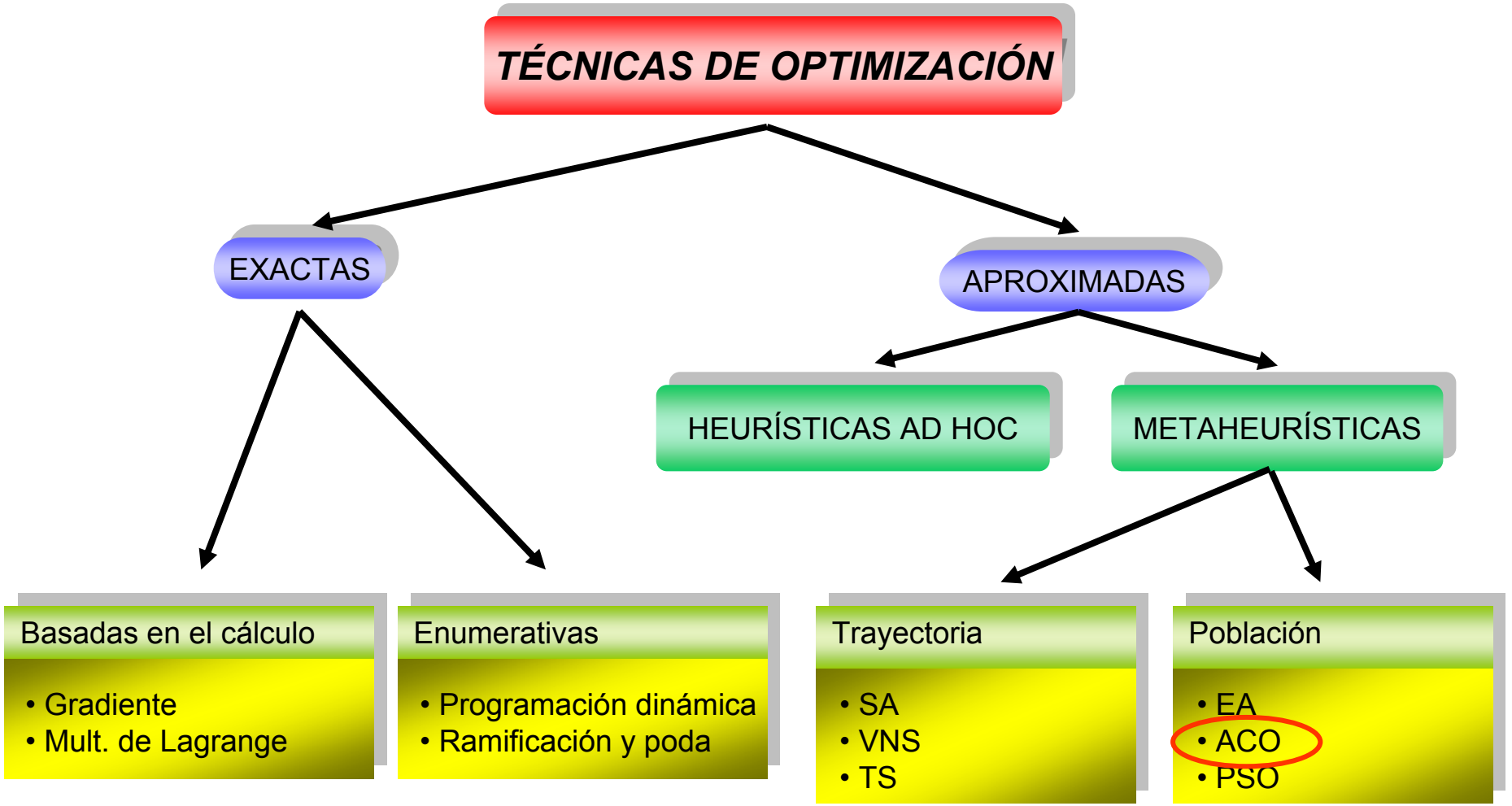
Model checking heurístico

- La búsqueda de errores se puede guiar usando información heurística



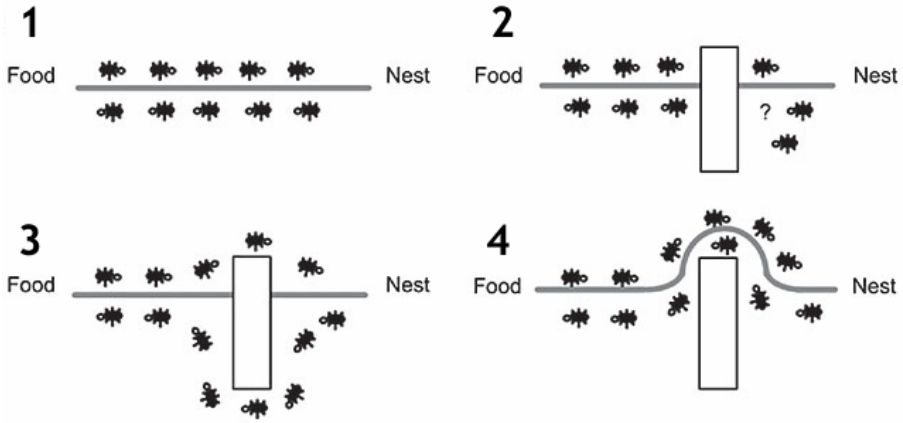
- Se han propuesto distintos tipos de funciones heurísticas:
  - Heurísticas basadas en fórmulas
  - Heurísticas estructurales
  - Heurísticas para interbloqueos
  - Heurísticas basadas en estados

Técnicas de optimización



ACO: Introducción

- **Los algoritmos basados en colonias de hormigas (ACO) están inspirados en el comportamiento de las hormigas reales cuando buscan comida**

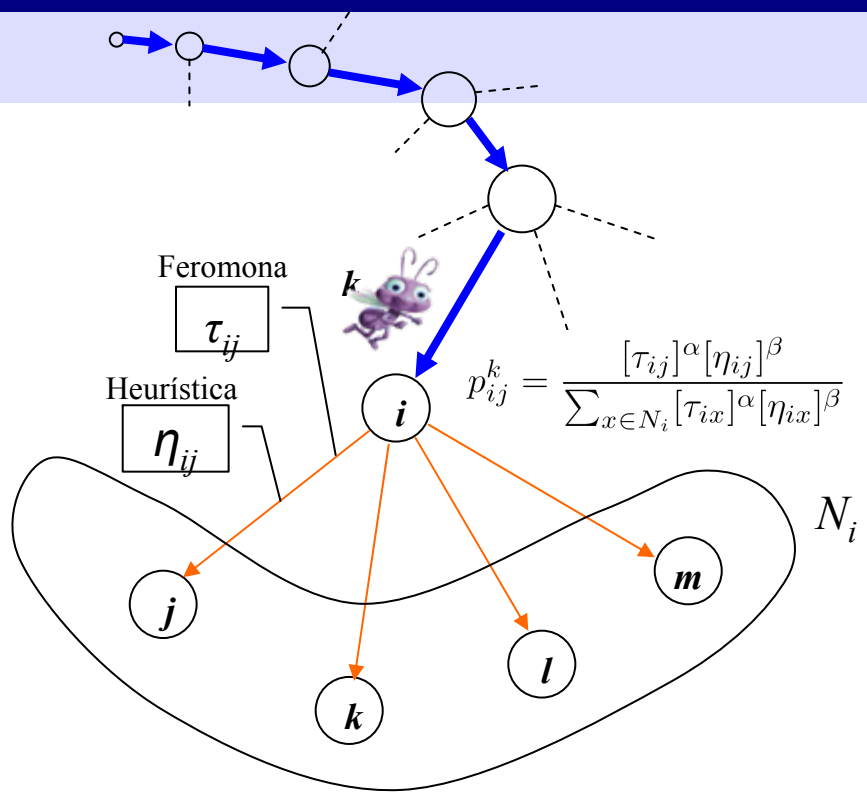


- **Pseudocódigo de un ACO**

```
procedure ACOMetaheuristic
    ScheduleActivities
        ConstructAntsSolutions
        UpdatePheromones
        DaemonActions // optional
    end ScheduleActivities
end procedure
```

ACO

- La hormiga escoge el siguiente nodo aleatoriamente
- La probabilidad de elegir un nodo depende del rastro de feromona y el valor heurístico del nodo o eje
- La hormiga se detiene cuando completa una solución

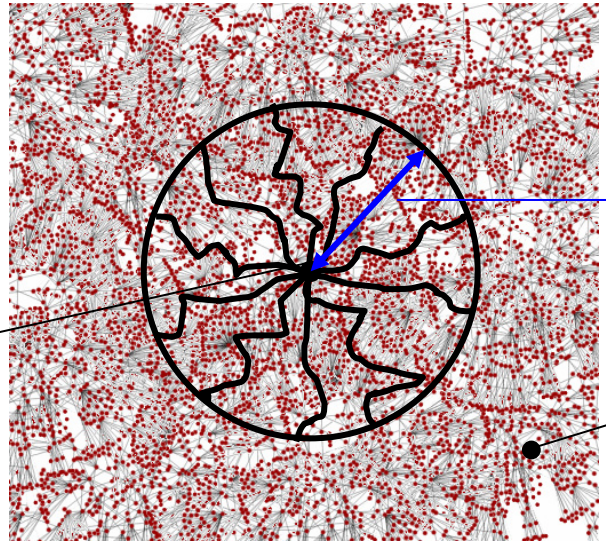


Actualización de feromona

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \Delta\tau_{ij}^{bs} \quad \text{con} \quad 0 \leq \rho \leq 1$$

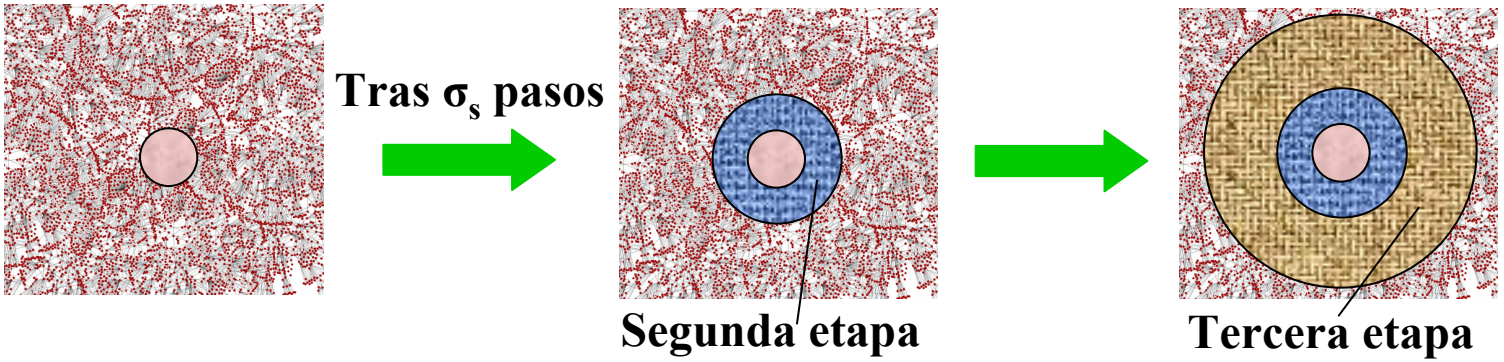
ACOhg: ACO for huge graphs

La longitud del camino de las hormigas se limita a λ_{ant}



Qué pasa si…?

Los nodos iniciales para la construcción de los caminos cambian



Programas y parámetros

- **Usamos 3 programas concurrentes escalables en Java**

Programa	LdC	Clases	Procesos
`dinj`	63	1	j+1
`phij`	176	3	j+1
`marj`	186	4	j+1

- **Parámetros de ACOhg**

Parámetro	*m*	*csize*	λ_{ant}	σ_s	ι	ρ	τ_0^{min}	τ_0^{max}	α	β	p
Valor	100	10	40	40	10	0.2	0.0	1.0	1.0	2.0	100

- **Heurística basada en interbloqueo**
- **100 ejecuciones independientes**
- **Pentium 4 a 2.8 GHz con 1 GB de RAM y Linux**

Programas y parámetros

- **Usamos 3 programas concurrentes escalables en Java**

Programa	LdC		Procesos
`dinj`	63	1	j+1
`phij`	176	1	j+1
`marj`	186	4	j+1

j=2 a 16

j=2 a 16

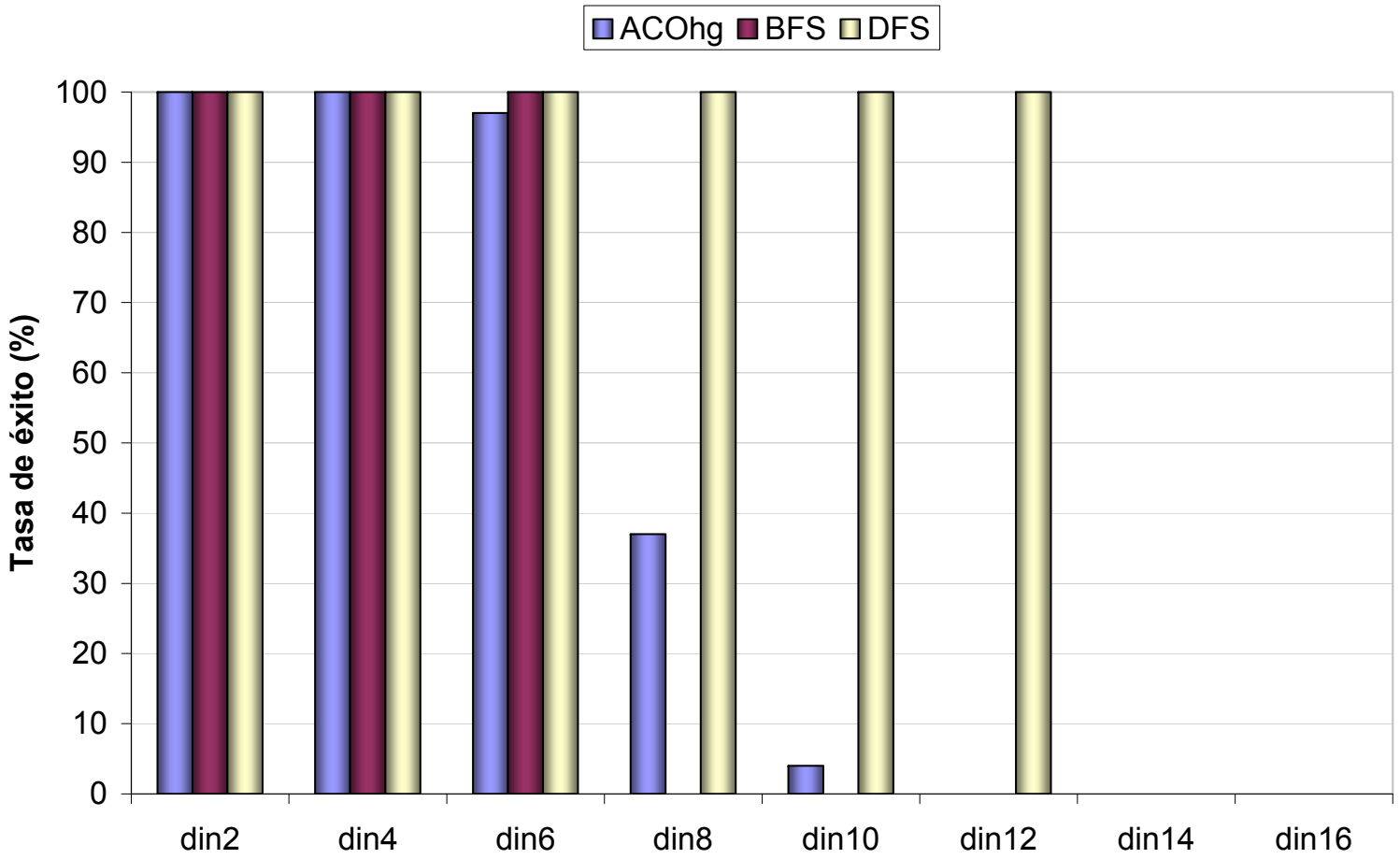
j=2 a 6

- **Parámetros de ACOhg**

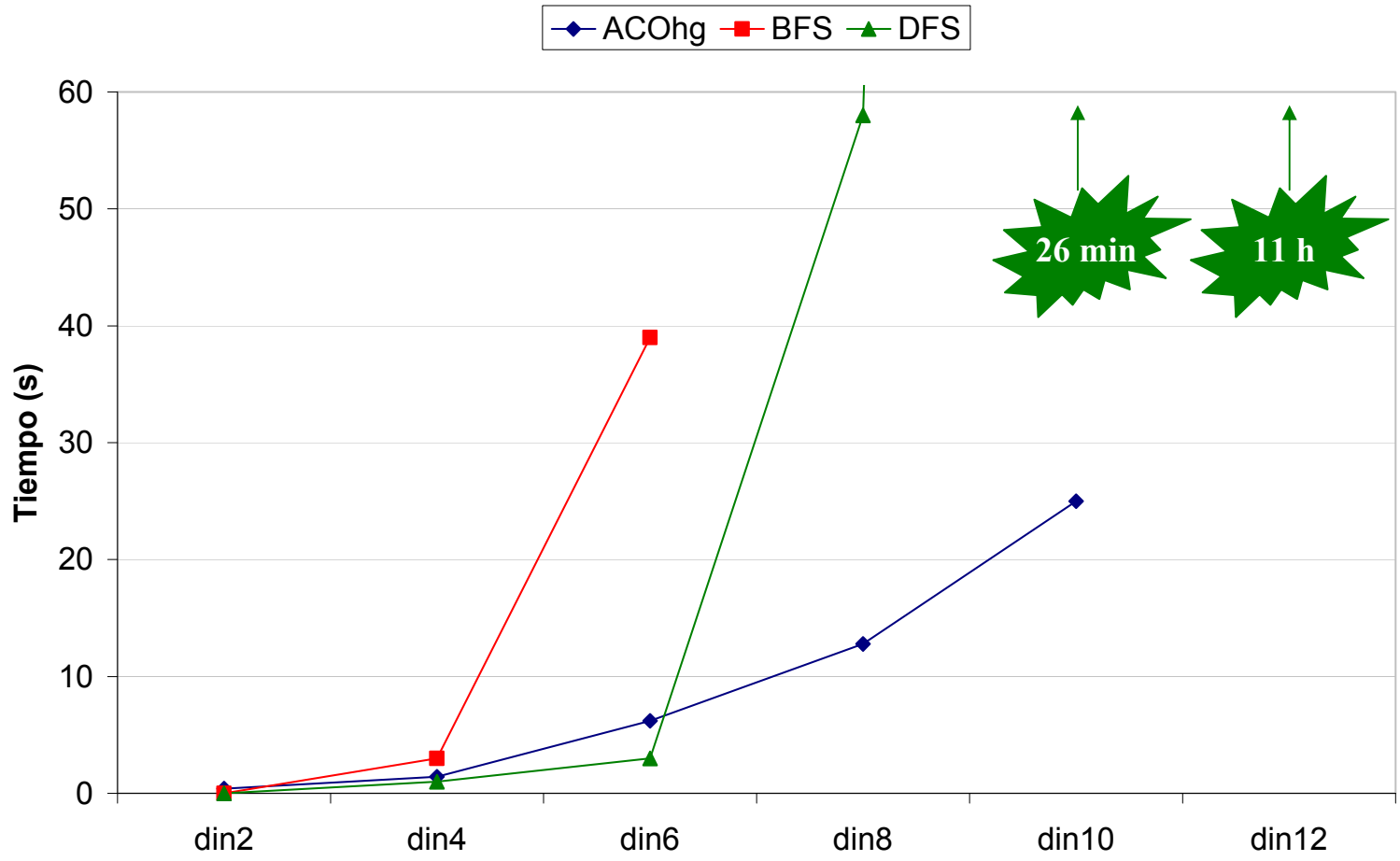
Parámetro	*m*	*csize*	λ_{ant}	σ_s	ι	ρ	τ_0^{min}	τ_0^{max}	α	β	p
Valor	100	10	40	40	10	0.2	0.0	1.0	1.0	2.0	100

- **Heurística basada en interbloqueo**
- **100 ejecuciones independientes**
- **Pentium 4 a 2.8 GHz con 1 GB de RAM y Linux**

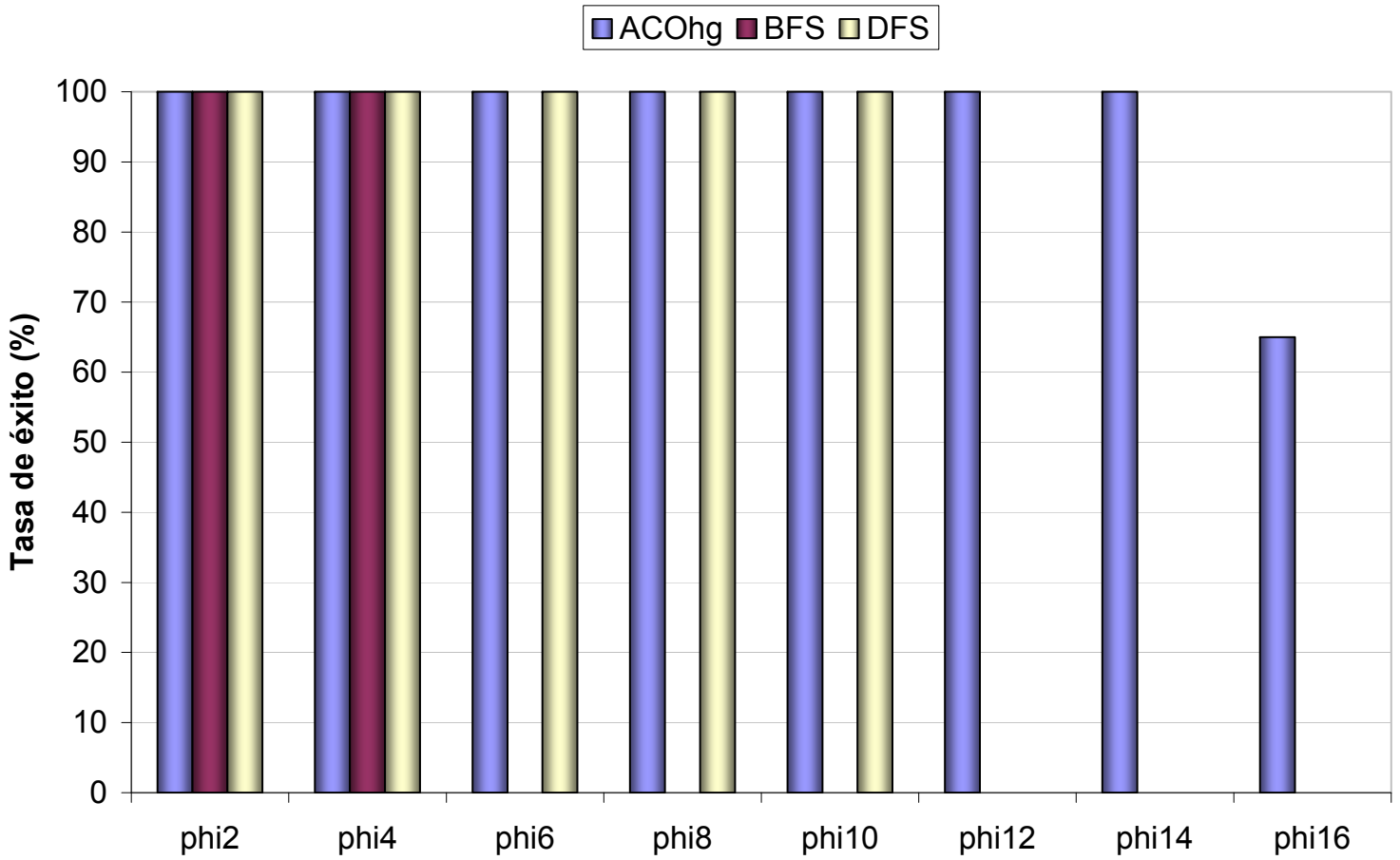
Resultados: din



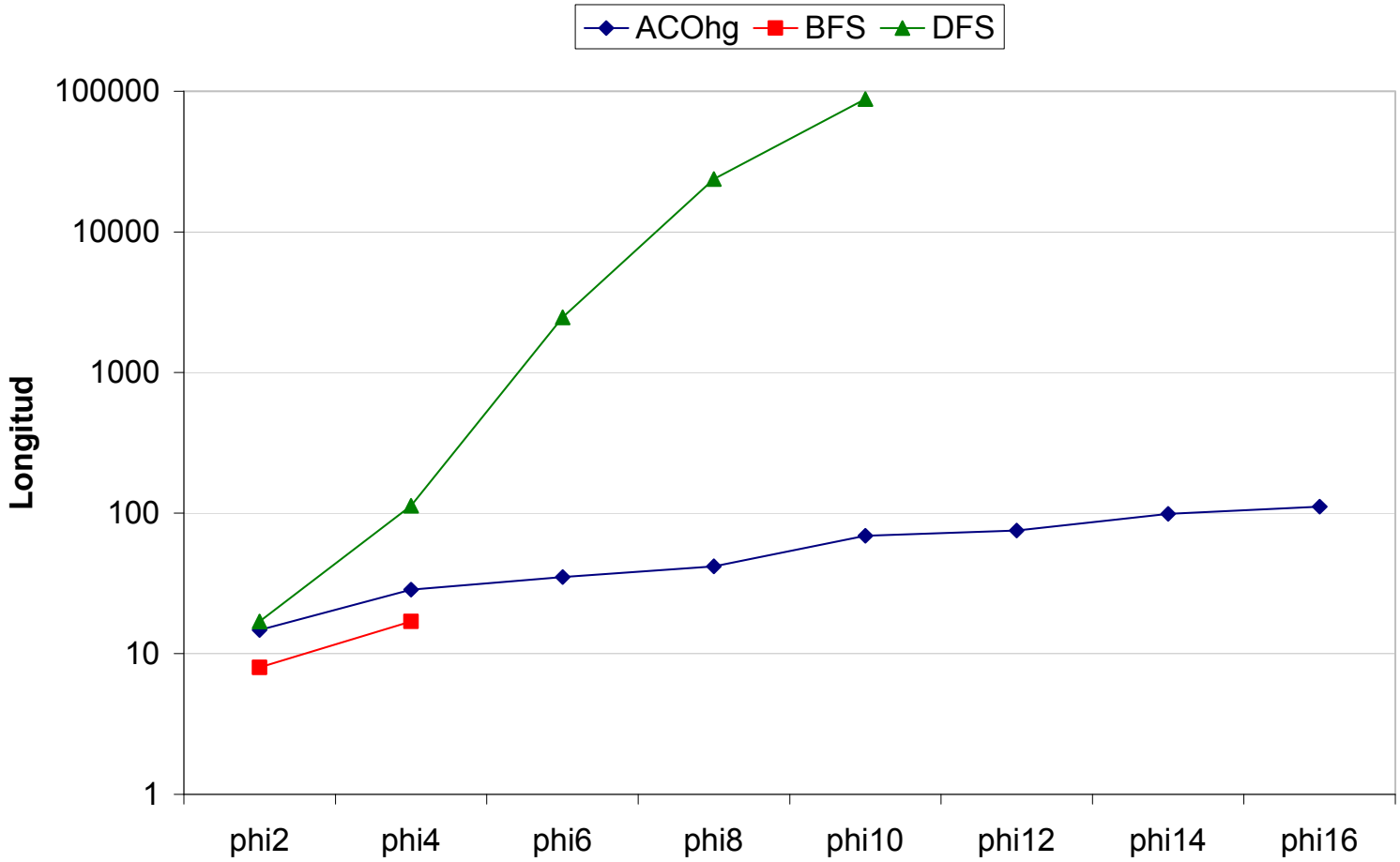
Resultados: din



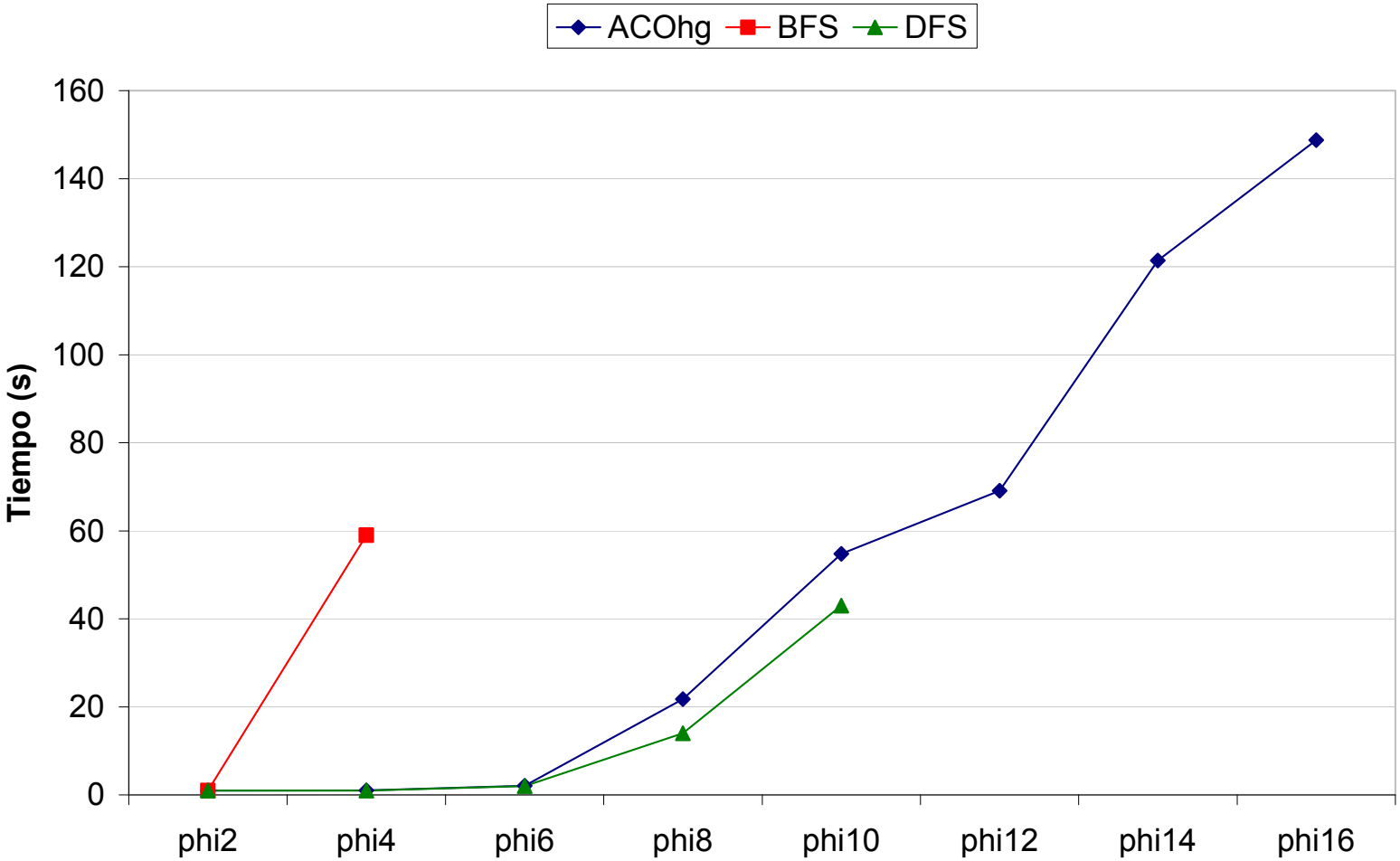
Resultados: phi



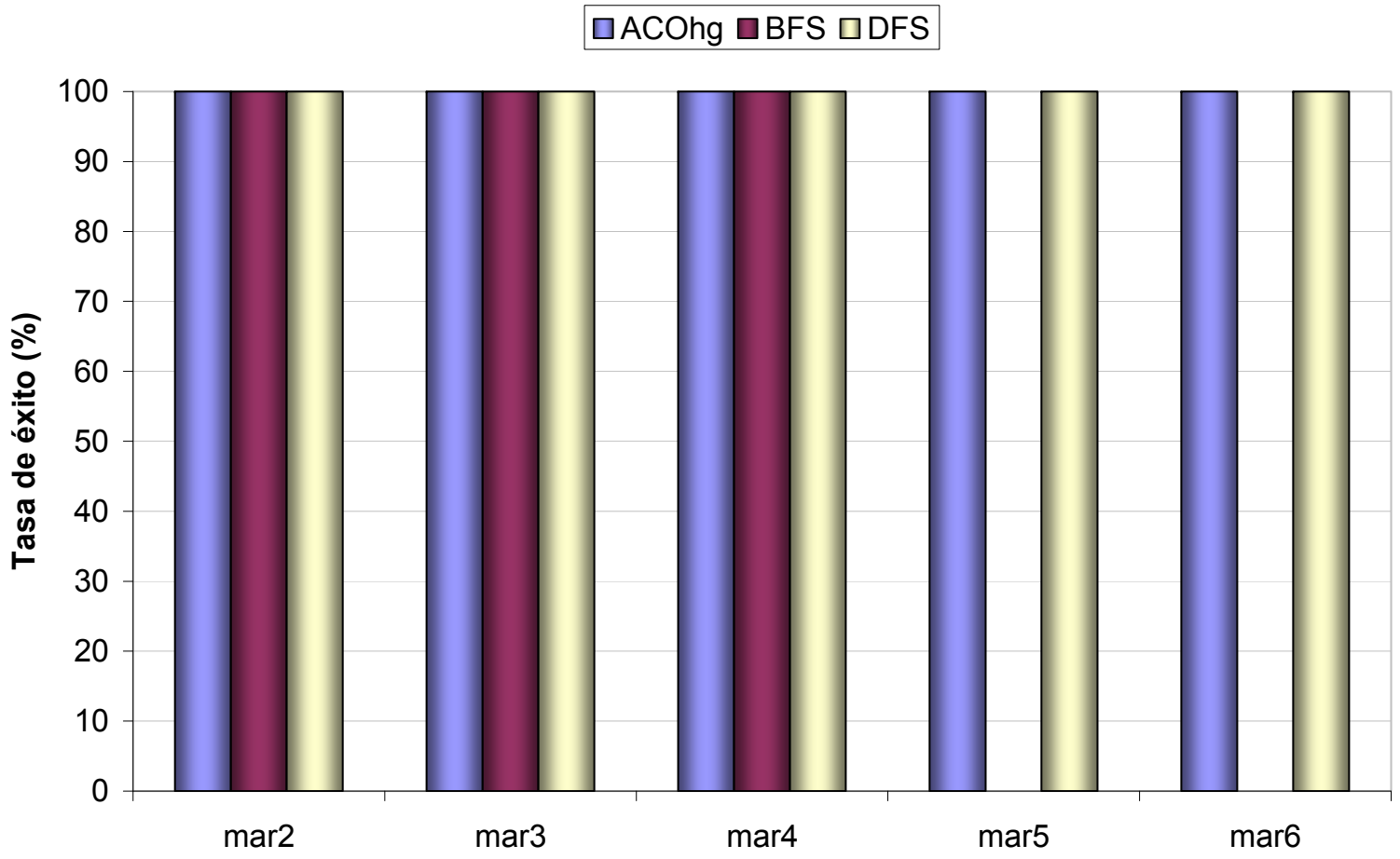
Resultados: phi



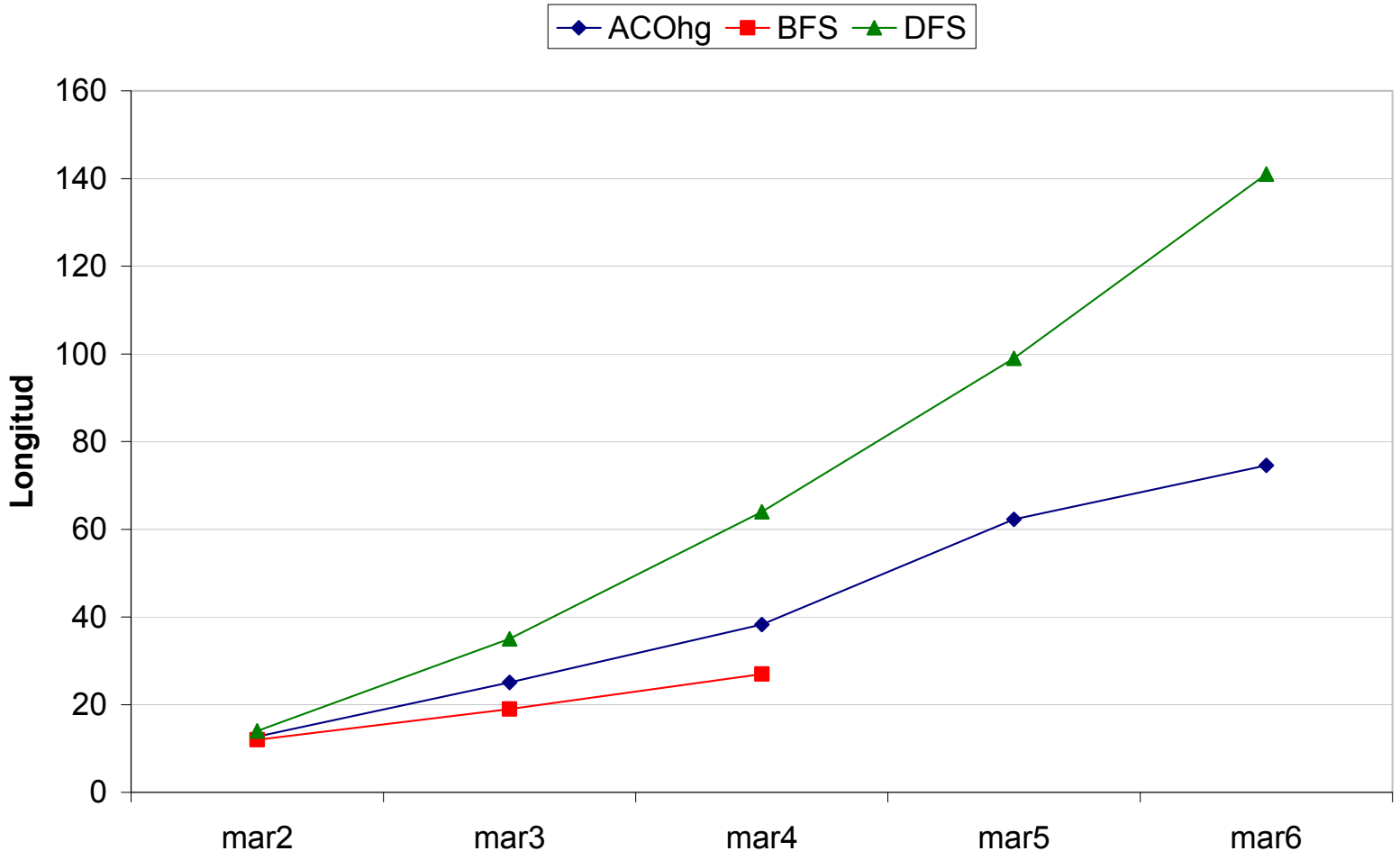
Resultados: phi



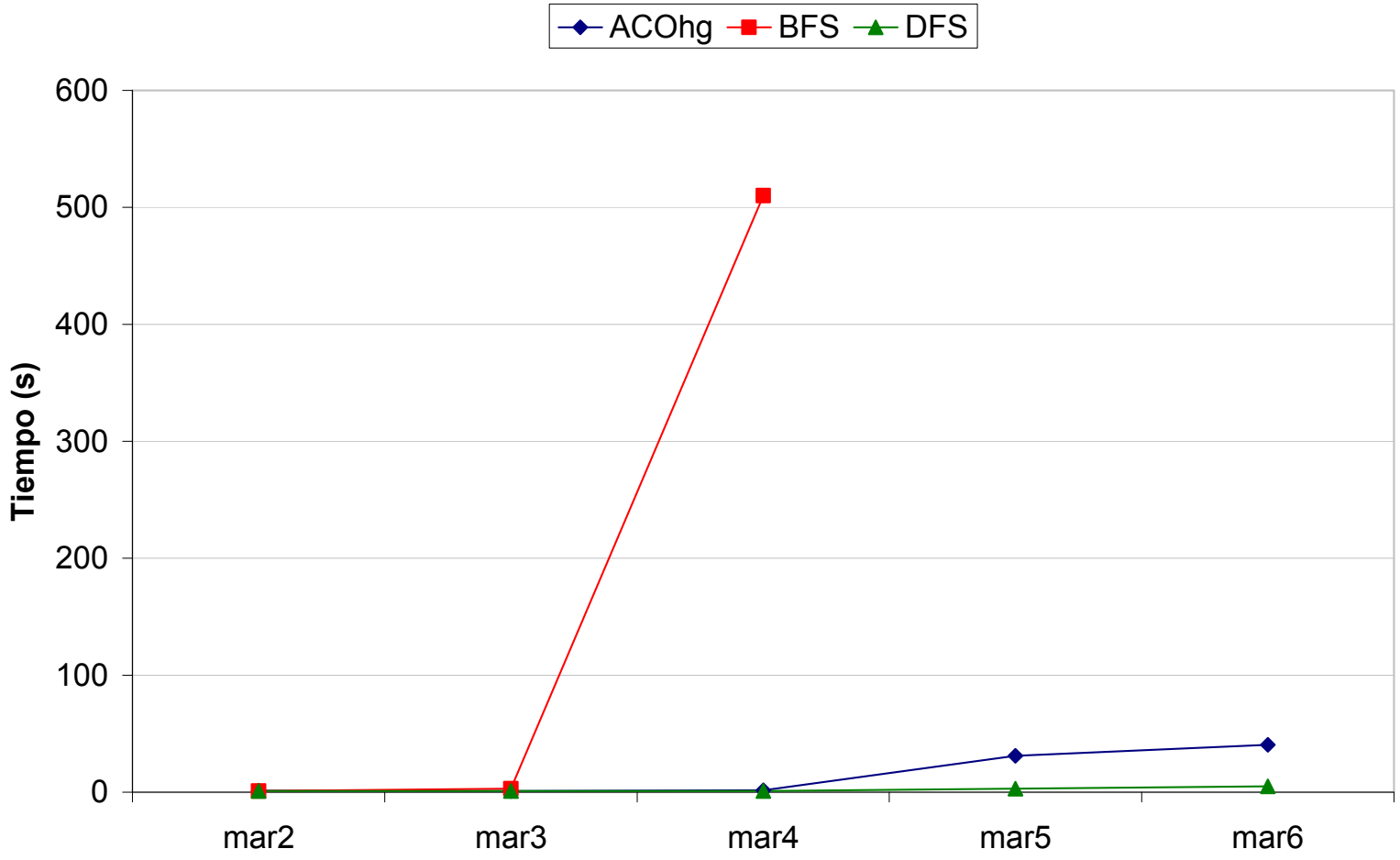
Resultados: mar



Resultados: mar



Resultados: mar



Conclusiones y trabajo futuro

Conclusiones

- ACOhg encuentra errores en ocasiones en las que BFS y DFS necesitan más memoria
- Las trazas de error obtenidas por ACOhg son cortas
- El tiempo de ejecución requerido por ACOhg es reducido

Trabajo futuro

- Búsqueda de violaciones de propiedades LTL
- Ejecución en paralelo de ACOhg para disponer de más memoria
- Combinar ACOhg con técnicas para reducir la memoria requerida para la búsqueda como reducción de orden parcial, simetría o ejecución delta

Gracias por su atención !!!