Ant Colony Optimization for Model Checking

LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD DE MÁLAGA

GISUM
Grupo de Ingeniería del Software de la Universidad de Málaga

Enrique Alba and Francisco Chicano

Introduction

• Nowadays software is very complex

• An error in a software system can imply the loss of lot of money …

… and even human lifes

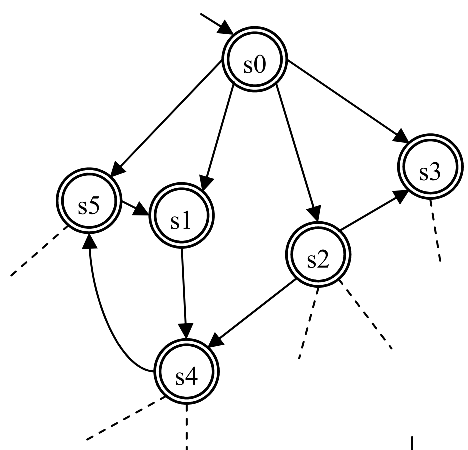
• Techniques for proving the correctness of the software are required

• Model checking → fully automatic

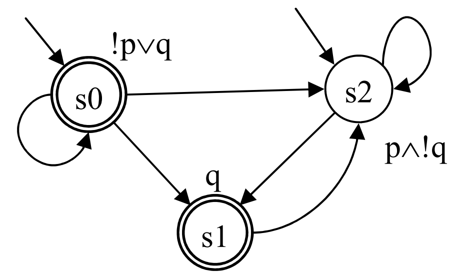
Explicit State Model Checking

- **Objective: Prove that model M satisfies the property f: $M \models f$**
- **SPIN: the property *f* is an LTL formula**

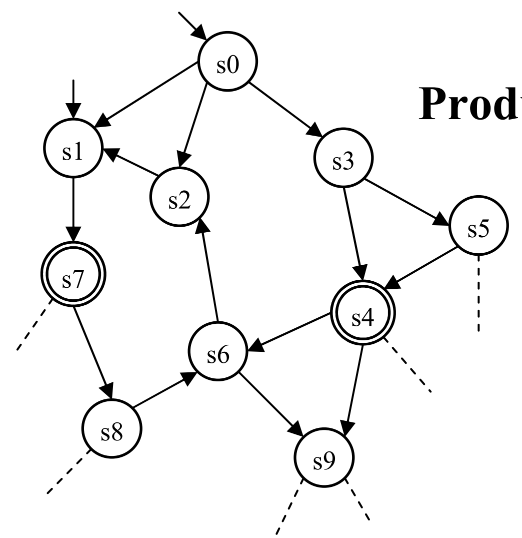
Model *M*



LTL formula ¬*f*



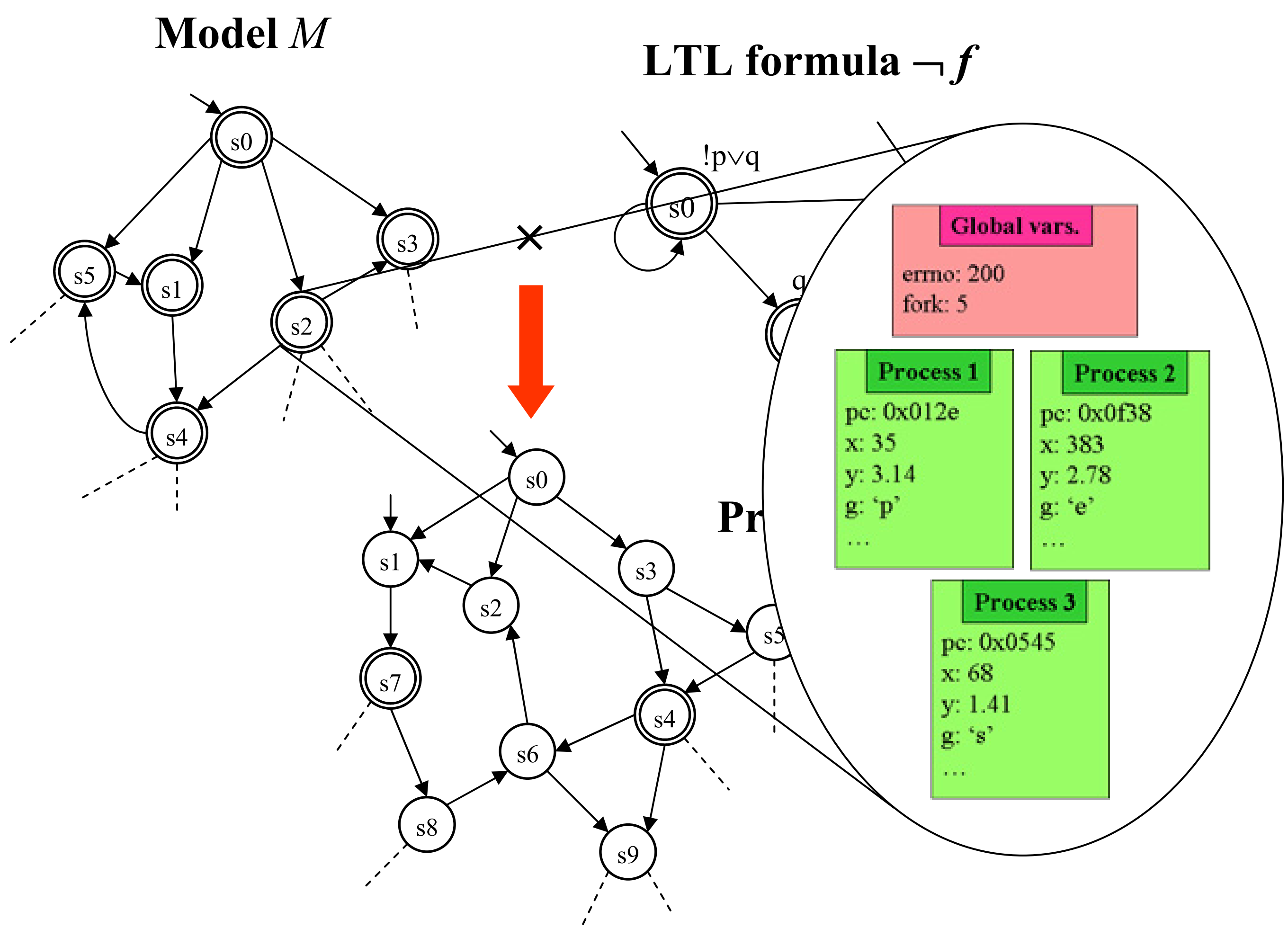
×



Product Büchi automaton

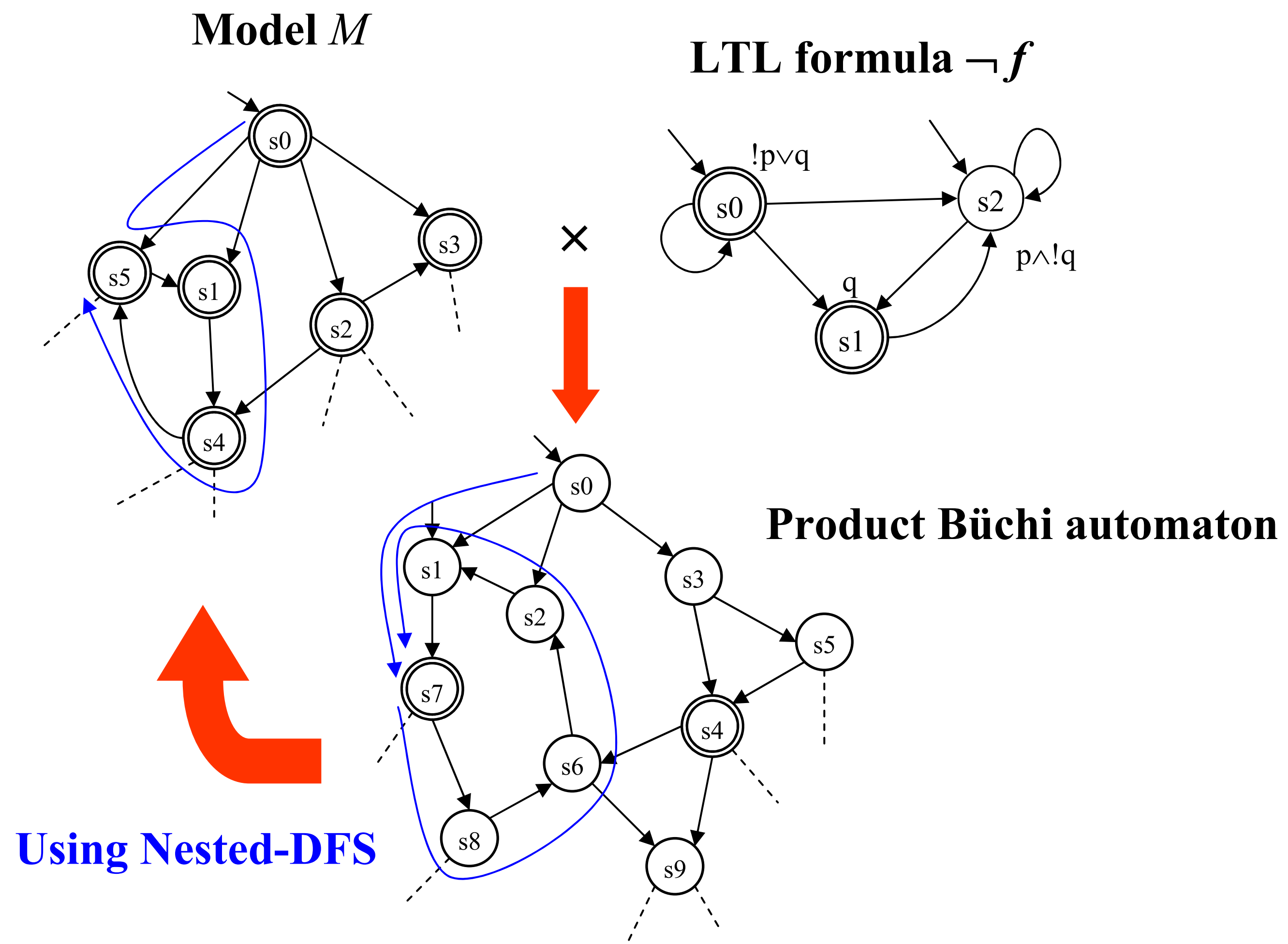
Explicit State Model Checking

- **Objective: Prove that model M satisfies the property f: $M \models f$**
- **SPIN: the property f is an LTL formula**



Explicit State Model Checking

- **Objective: Prove that model M satisfies the property f: $M \models f$**
- **SPIN: the property *f* is an LTL formula**



Safety Properties

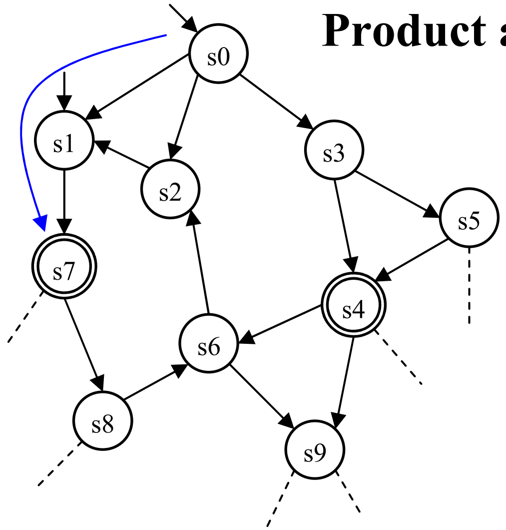
- Safety properties are those expressed by an LTL formula of the form:

$$\boxed{f = \Box p}$$

where p is a past formula

- Finding one counterexample ≡ finding one accepting state

Product automaton

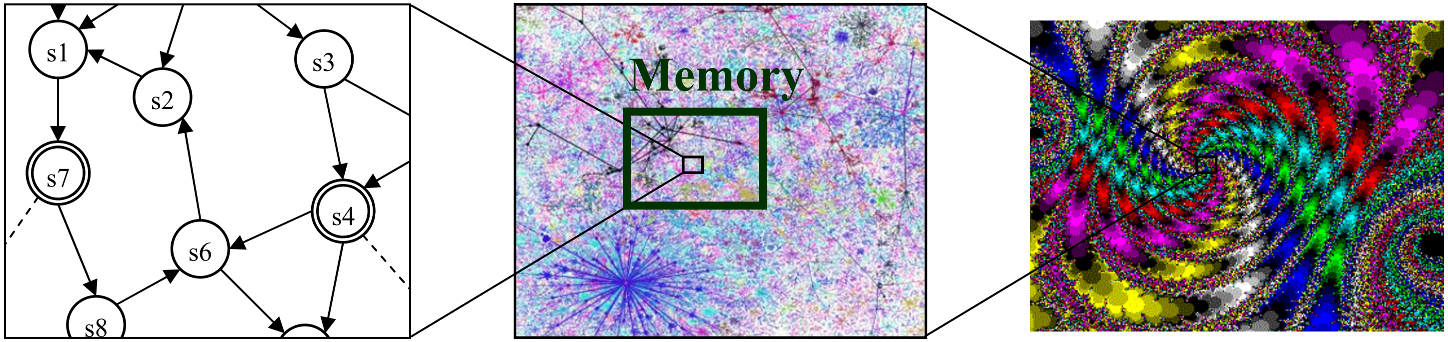


Safety Properties
Deadlocks
Invariants
Assertions
…

- Classical algorithms for graph exploration can be used: DFS and BFS

State Explosion Problem

- Number of states very large even for small models



- Example: Dining philosophers with *n* philosophers → 3^n states

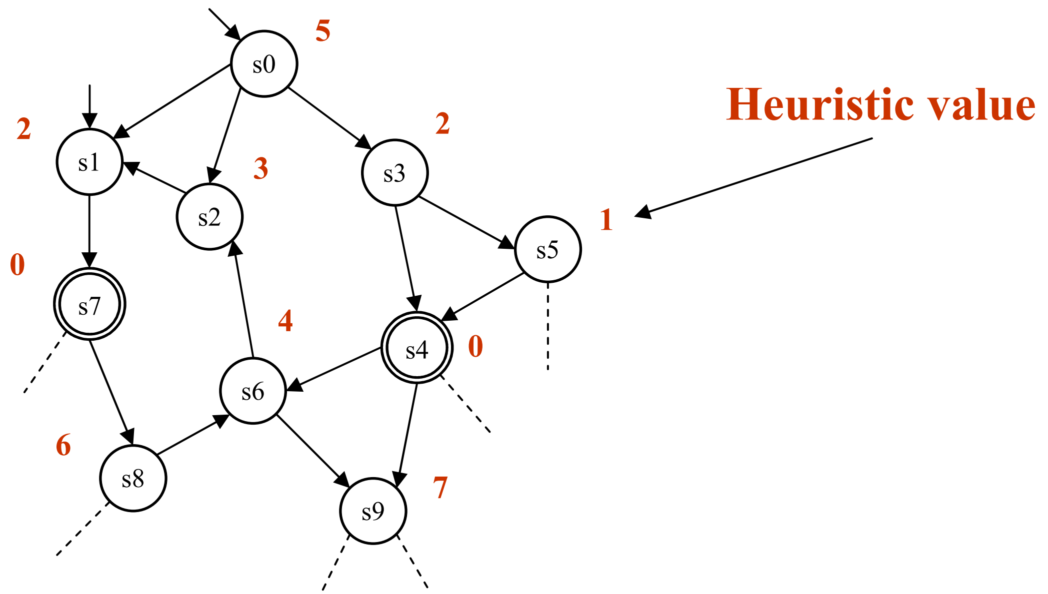
  20 philosophers → 1039 GB for storing the states

- Solutions: state compression, bitstate hashing, partial order reduction, symmetry reduction, symbolic model checking

- Large models cannot be verified but errors can be found

Heuristic Model Checking

- The search for errors can be directed by heuristics using algorithms like A*, IDA*, WA*, Best-First, and so on



- Different kinds of heuristic functions have been proposed in the past:
  - Formula-based heuristics: depends on the LTL formula
  - Structural heuristics: code coverage
  - Deadlock-detection heuristics: active process
  - State-dependent heuristics: hamming distance

Introduction

Background

Ant Colony Optimization

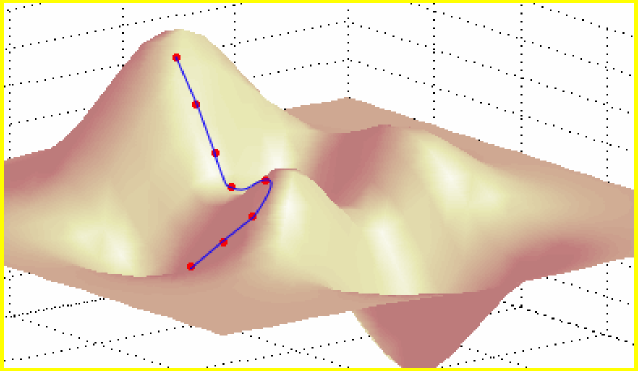
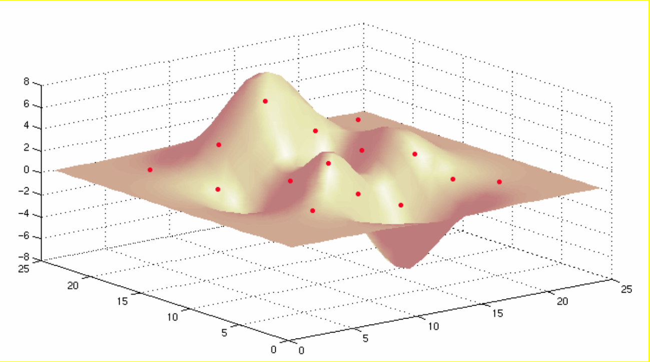
Experiments

Conclusions & Future Work

Metaheuristic Algorithms

- Designed to solve optimization problems
  - ➢ Maximize or minimize a given function: the fitness function
- They can find "good" solutions with "reasonable" resources

Metaheuristic Algorithms

Single solution	Population
	

Metaheuristic Algorithms

Single solution

- Greedy Randomized Adaptive Search Procedure
- Iterated Local Search
- Variable Neighborhood Search
- Tabu Search
- Simulated Annealing
- Iterative Improvement
- Guided Local Search

Population

- Estimation of Distribution Algorithms
- Evolutionary Computation
- Scatter Search
- Ant Colony Optimization
- Particle Swarm Optimization

Metaheuristic Algorithms

Single solution

- Greedy Randomized Adaptive Search Procedure
- Iterated Local Search
- Variable Neighborhood Search
- Tabu Search
- Simulated Annealing

Genetic Algorithms

Alba & Troya, 1996

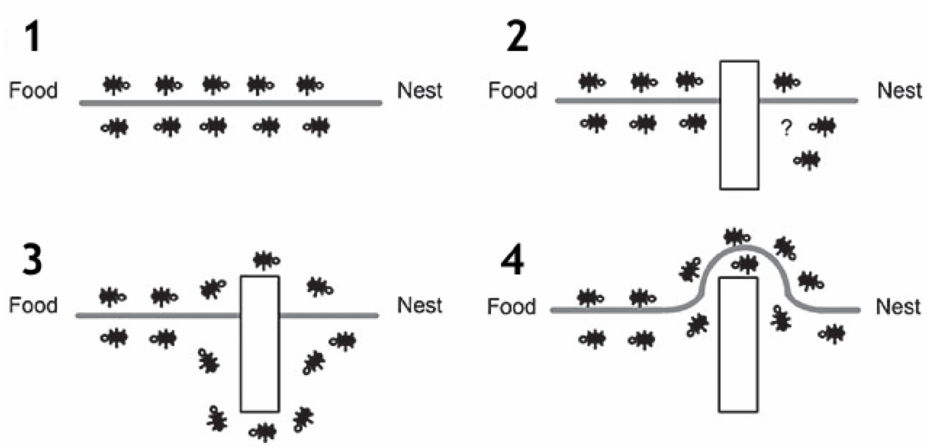
Godefroid & Khurshid, 2002, 2004

Population

- Estimation of Distribution Algorithms
- Evolutionary Computation
- Scatter Search
- Ant Colony Optimization
- Particle Swarm Optimization

Ant Colony Optimization

- Ant Colony Optimization (ACO) metaheuristic is inspired in the foraging behavior of the real ants

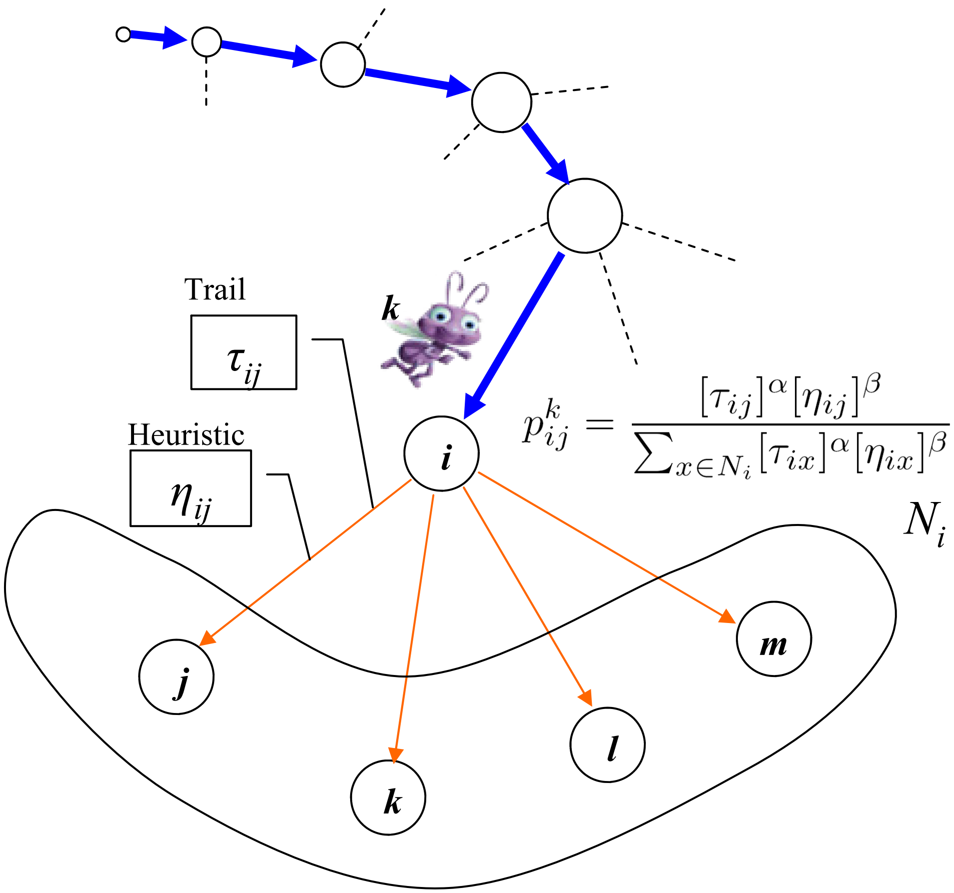


- ACO Pseudo-code

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

Ant Colony Optimization

- **Construction Phase**



$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{x \in N_i}[\tau_{ix}]^{\alpha}[\eta_{ix}]^{\beta}}$$

Ant Colony Optimization

- **Pheromone update**
  - **During the construction phase**

$$\tau_{ij} \leftarrow (1 - x_i)\tau_{ij} \quad \text{with} \quad 0 \leq x_i \leq 1$$

  - **After the construction phase**

$$\tau_{ij} \leftarrow \rho\tau_{ij} + \Delta\tau_{ij}^{bs} \text{ with} \quad 0 \leq \rho \leq 1$$

- **Trail limits (particular of *MM*AS)**
  - **Pheromones are kept in the interval [τ_{min}, τ_{max}]**

$$\tau_{max} = \frac{Q}{1 - \rho}$$

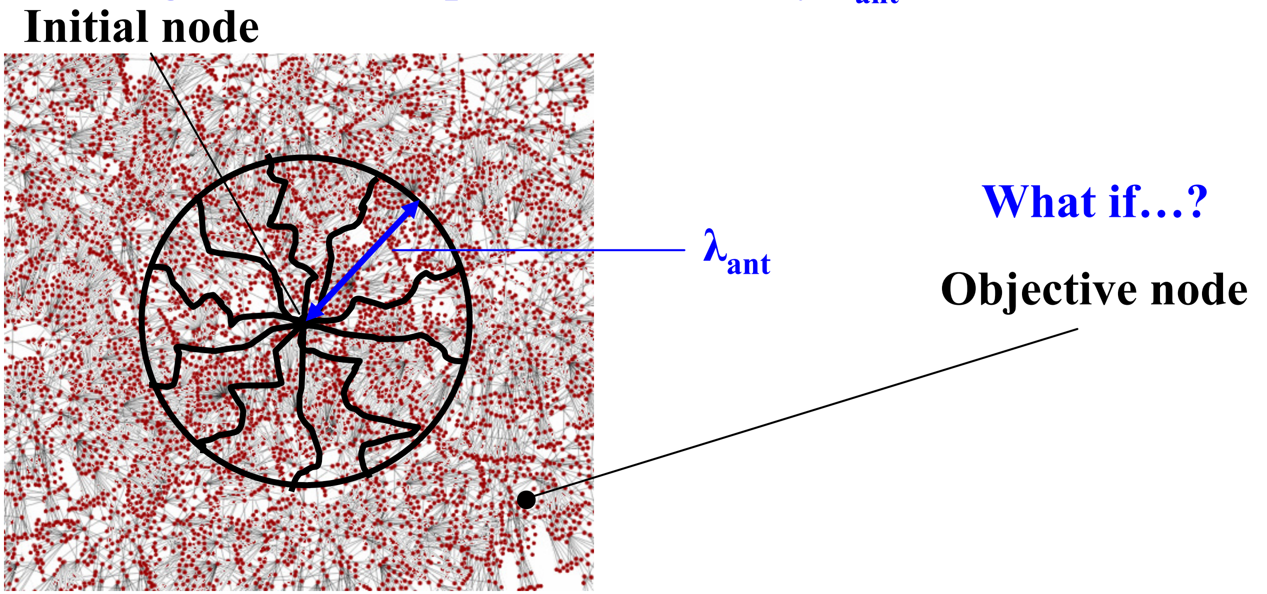
$$\tau_{min} = \frac{\tau_{max}}{a}$$

ACOhg: Motivation

- Existing ACO models cannot be applied to the search for errors in concurrent programs
  - The graph is very large, the construction of a complete solution could require too much time and memory
  - In some models the number of nodes of the graph is used for computing the initial pheromone values
- We need a new model for tackling these problems: ACOhg (ACO for Huge Graphs)
  - Constructs the ant paths and updates the pheromone values in the same way as the traditional models
  - Allows the construction of partial solutions
  - Allows the exploration of the graph using a bounded amount of memory
  - The pheromone matrix is never completely stored

ACOhg: Ant Paths Length

- The length of the ant paths is limited by λ_{ant}

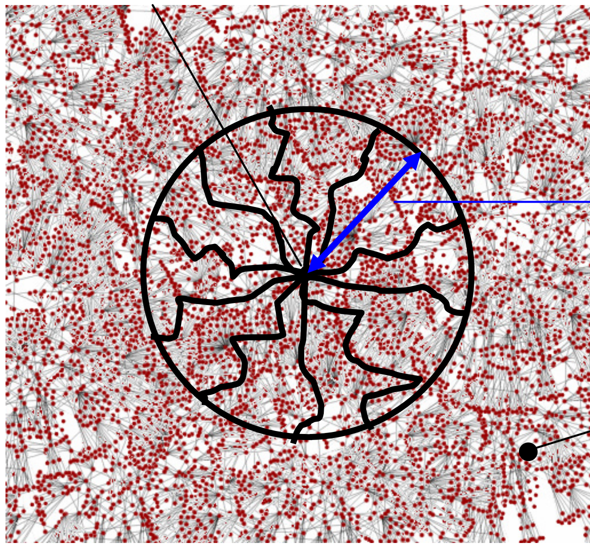


What if…?

ACOhg: Ant Paths Length

- The length of the ant paths is limited by λ_{ant}

Initial node



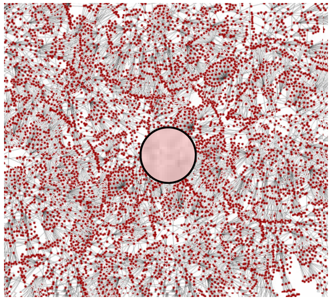
λ_{ant}

What if…?

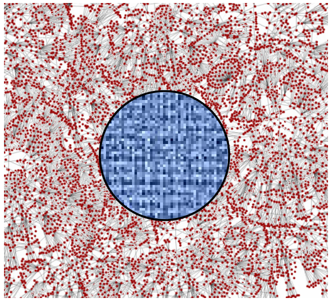
Objective node

Two alternatives

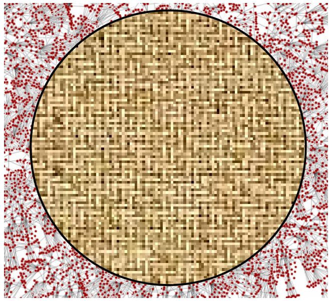
Expansion Technique: λ_{ant} changes



After σ_i steps

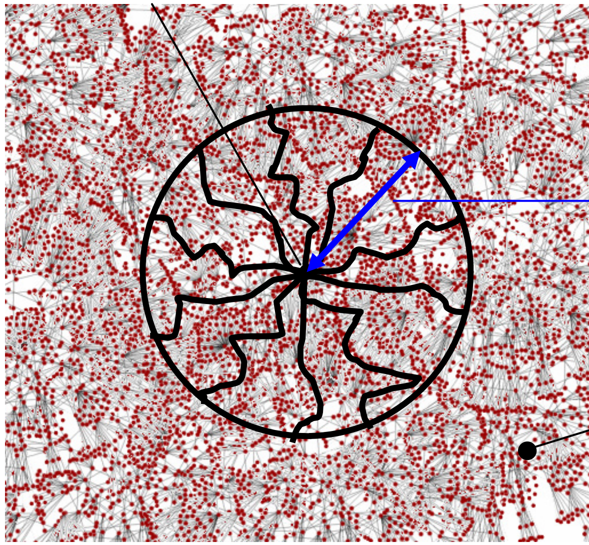


$\lambda_{ant} = \lambda_{ant} + \delta_l$



ACOhg: Ant Paths Length

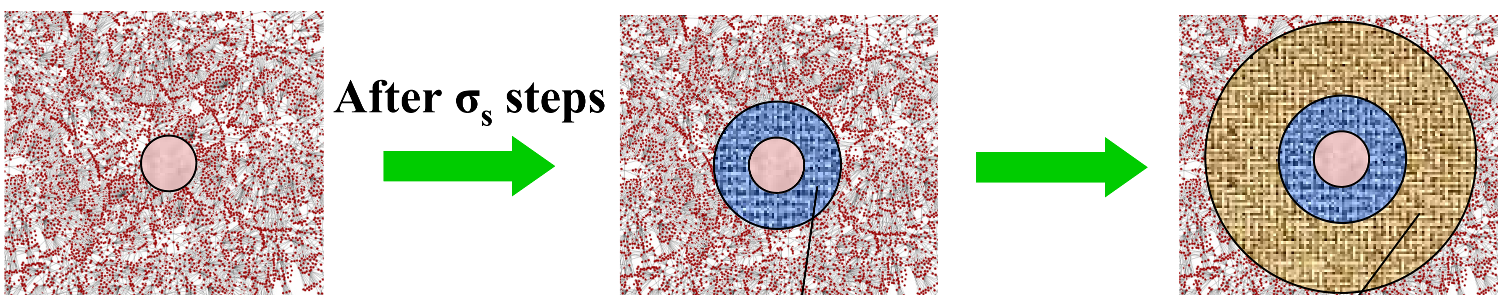
- The length of the ant paths is limited by λ_{ant}



What if…?

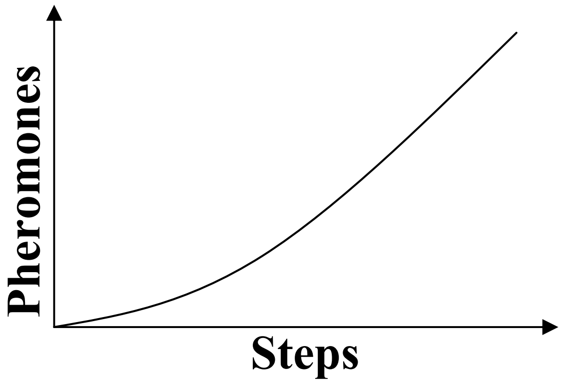
Two alternatives

Missionary Technique: starting nodes for path construction change

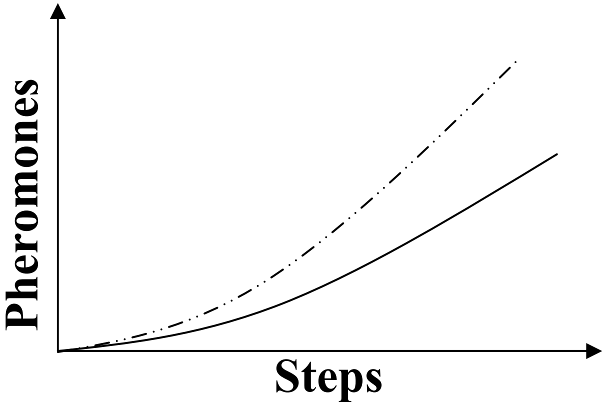


ACOhg: Pheromones

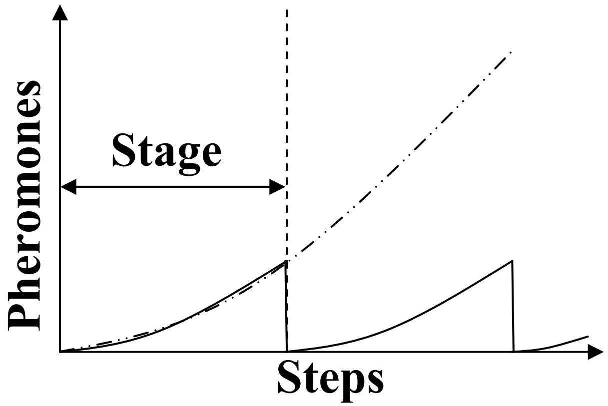
- The number of pheromone trails increases during the search
- This leads to memory problems
- We must remove some pheromone trails from memory



Remove pheromone trails τ_{ij} below a given threshold τ_θ

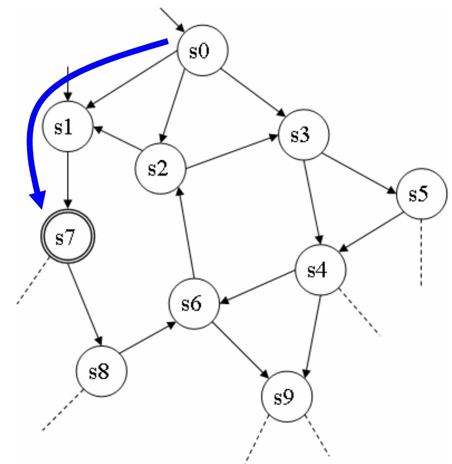


In the missionary technique, remove all pheromone trails after one stage



ACOhg: Fitness Function

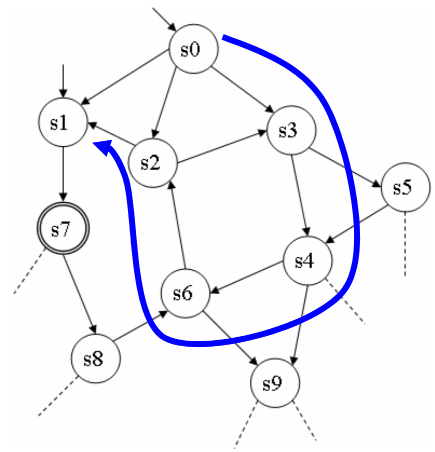
- **The fitness function must be able to evaluate partial solutions**
- **Penalties are added for partial solutions and solutions with cycles**



Complete solution

$$p = 0$$

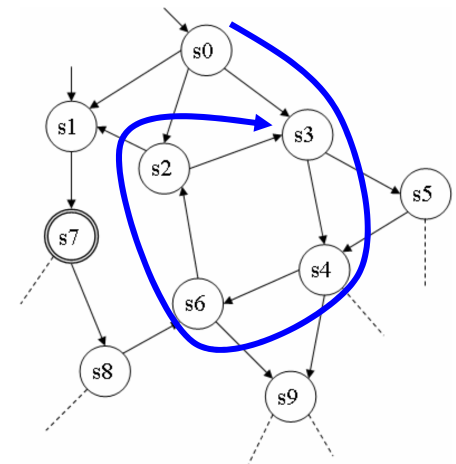
Total penalty



Partial solution without cycle

$$p = p_p$$

Penalty constant for partial solutions



Partial solution with cycle

$$p = p_p + p_c \frac{\lambda_{ant} - l}{\lambda_{ant} - 1}$$

Penalty constant for solutions with cycles

Path length

Experiments: Models

- **We selected 5 Promela models for the experiments**

Model	LoC	States	Processes	Safety Property
basic_call2	198	33667	3	deadlock
garp	272	*unknown*	8	deadlock
giop12	717	27877	10	deadlock
marriers4	142	*unkown*	5	deadlock
phi8	34	6561	9	deadlock

- **For `marriers4` and `garp` the states do not fit into the main memory of the computer**

Experiments: Parameters

- **The ACOhg model was implemented inside the MALLBA library and and then included into the HSF-SPIN model checker**

Parameter	Value
Steps	10
Colony size	10
λ_{ant}	10
x_i	0.0
ρ	0.9
α	1.0
β	0.0/1.0

- **These parameters belong to a Min-Max Ant System (*MM*AS)**
- **Fitness function: length of the path + penalty**
- **Two variants: using no heuristic (*MM*AS-b) and using it (*MM*AS-h)**
- **Machine: Pentium 4 at 2.8 GHz with 512 MB**

Experiments: Comparison

- We compare *MM*AS-b and *MM*AS-h against DFS, BFS, and A*

Concurrent System		DFS	BFS	$\mathcal{MM}$AS-b	A*	$\mathcal{MM}$AS-h
`basic_call2`	Len.	82.00	**26.00**	30.24	**26.00**	31.30
Basic call protocol	CPU (ms)	**10.00**	80.00	41.70	110.00	**49.40**
with 2 users	Mem. (KB)	**2713.00**	16384.00	4219.44	17408.00	**4278.44**
`garp`	Len.	65.00	**17.00**	24.70	**17.00**	24.00
Generic Attribute	CPU (ms)	10.00	53180.00	**6.00**	2820.00	**9.50**
Registration Protocol	Mem. (KB)	3357.00	480256.00	**2532.28**	122880.00	**2634.44**
`giop12`	Len.	48.00	**43.00**	43.45	**43.00**	**43.00**
CORBA General Inter-ORB	CPU (ms)	**10.00**	350.00	41.20	490.00	**26.80**
Protocol (1 client, 2 servers)	Mem. (KB)	**2901.00**	49152.00	4085.12	37888.00	**3202.24**
`marriers4`	Len.	-	-	**85.79**	-	**88.76**
Stable marriage problem	CPU (ms)	-	-	**148.60**	-	**91.90**
with 4 suitors	Mem. (KB)	-	-	**15388.39**	-	**10625.80**
`phi8`	Len.	1338.00	**10.00**	**10.00**	**10.00**	**10.00**
Dining philosophers	CPU (ms)	70.00	60.00	**21.90**	0.00	35.20
with 8 philosophers	Mem. (KB)	29696.00	17408.00	**4333.92**	**2105.00**	4986.68

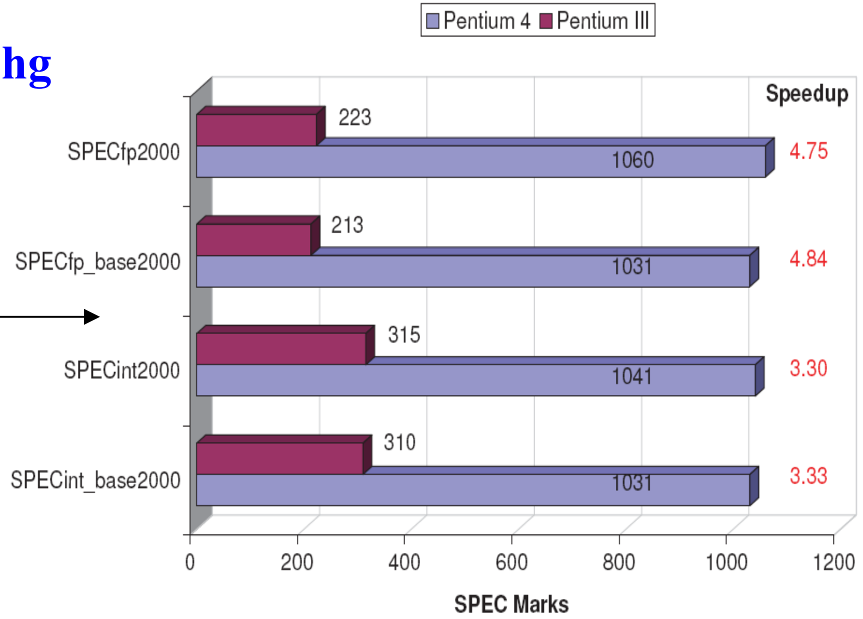
- *MM*AS algorihtms are the only ones able to find errors in `marriers4`
- *MM*AS algorithms require less memory than BFS
- *MM*AS algorithms get shorter (better) error trails than DFS
- We conclude that *MM*AS algorithms are the best trade-off between efficacy and efficiency (good results with low resources)

Experiments: ACOhg vs. GA

- **GA is the previous metaheuristic algorithm applied to this problem**
- **We use `phi17` and `needham` for the comparison (Godefroid & Khurshid, 2002)**

Model	Algorithm	Hit (%)	Time (s)	Mem. (KB)
phi17	GA	52	197.00	n/a
	ACOhg-h	**100**	**0.28**	11274
needham	GA	3	3068.00	n/a
	ACOhg-h	**100**	**0.23**	4865

- **The results state that ACOhg has higher efficacy and efficiency than GA (even taking into account the differences in the machines)**



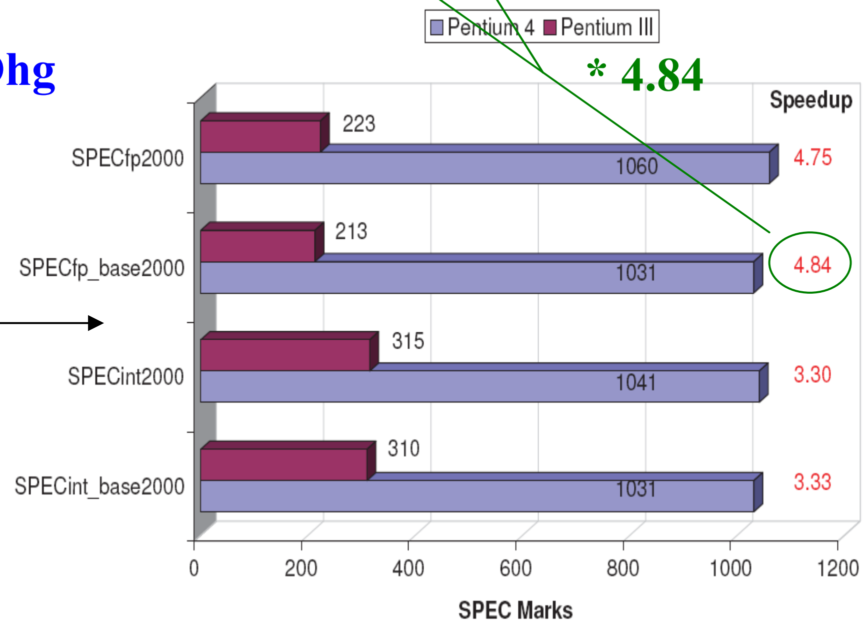
Experiments: ACOhg vs. GA

- **GA is the previous metaheuristic algorithm applied to this problem**
- **We use `phi17` and `needham` for the comparison (Godefroid & Khurshid, 2002)**

Model	Algorithm	Hit (%)	Time (s)	Mem. (KB)
phi17	GA	52	197.00	n/a
	ACOhg-h	**100**	**1.36**	11274
needham	GA	3	3068.00	n/a
	ACOhg-h	**100**	**1.11**	4865

* 4.84

- **The results state that ACOhg has higher efficacy and efficiency than GA (even taking into account the differences in the machines)**



Conclusions and Future Work

Conclusions

- ACOhg is able to outperform state-of-the-art algorithms used nowadays in current model checkers for finding errors
- The results obtained with ACOhg improve by far those obtained by GAs in the past
- This represents a promising starting point for the use of metaheuristic algorithms in model checking

Future Work

- We plan to study ACOhg algorithms in depth, exploring all the alternatives mentioned for the algorithm
- ACOhg algorithms can be combined with other techniques for reducing the amount of memory: e.g., Partial Order Reduction
- ACOhg can be extended to work in parallel and profit from the use of clusters of machines (parallel model checkers)

The End

Thanks for your attention !!!