GECCO 2007

ACOhg: Dealing with Huge Graphs

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 y Francisco Chicano

2007

Introduction

ACOhg

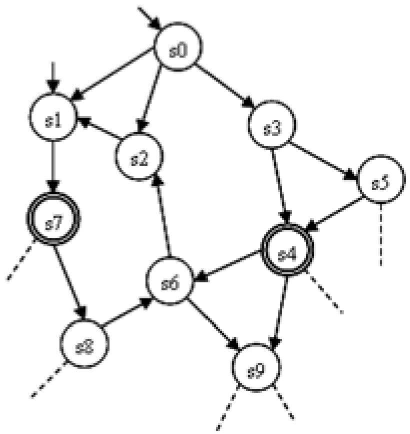
Systems Verification

Experiments

Conclusions & Future Work

Introduction

- ACO algorithms solve problems that can be translated into a **shortest path search** in a graph
- There exist problems in which the construction graph is **unknown and/or very large**



$$\frac{\alpha\vee\beta,\ \neg\beta\vee\gamma}{\alpha\vee\gamma}$$

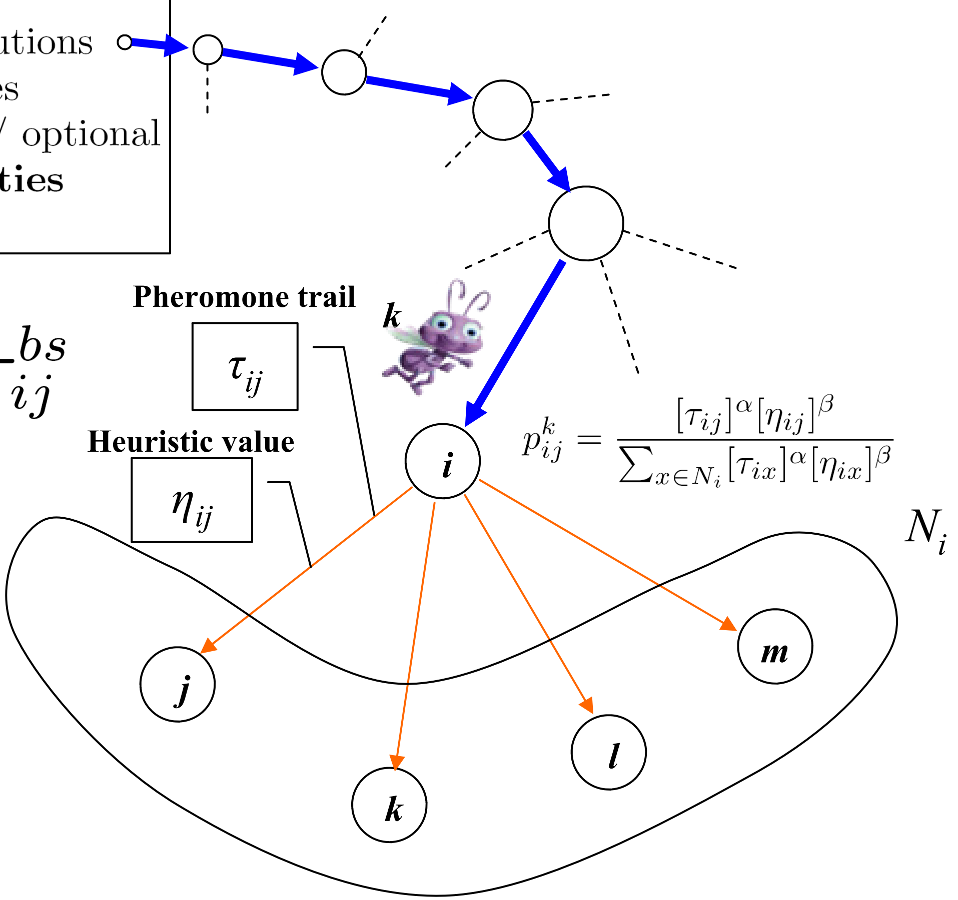
- Current ACO models **cannot be applied** to these problems
  - ➢ In a really large graph, the construction of a complete solution could require **too much time and memory**
  - ➢ In some models **the number of nodes** of the graph **is used** for computing the initial pheromone values

Introduction

- We need a **new model** for tackling these problems: **ACOhg** (ACO for Huge Graphs)

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

$$\tau_{ij} \leftarrow \rho\tau_{ij} + \Delta\tau_{ij}^{bs}$$

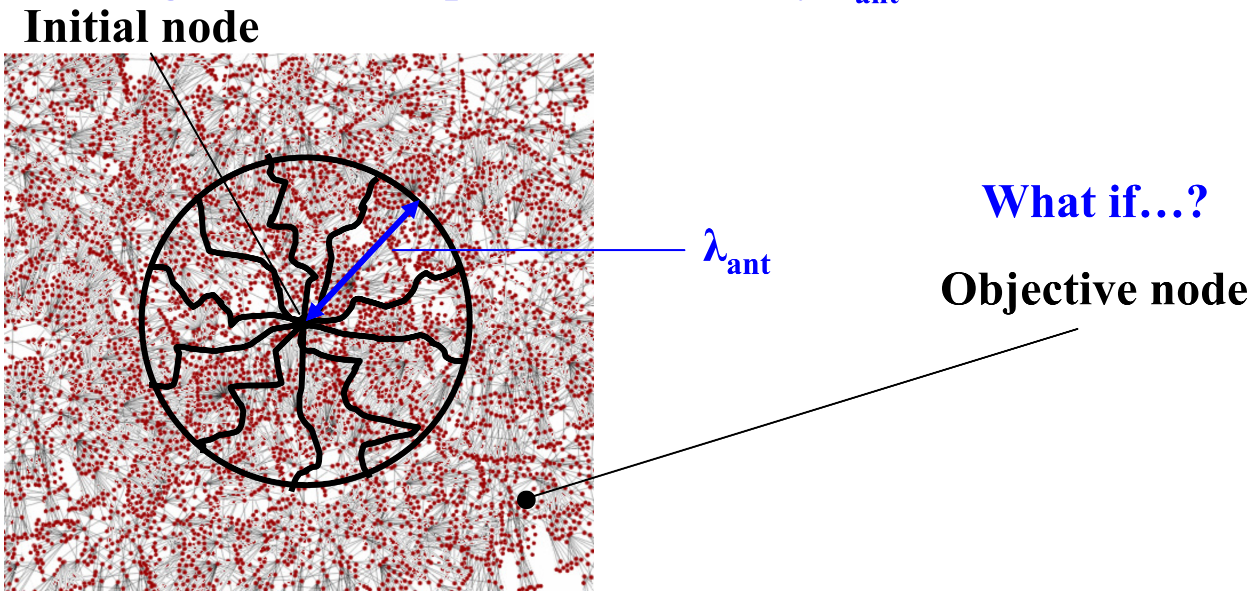


ACOhg extends current models with new ideas, but it performs the **construction phase** and the **pheromone update** in the same way

GECCO 2007

ACOhg: Ant Paths Length

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

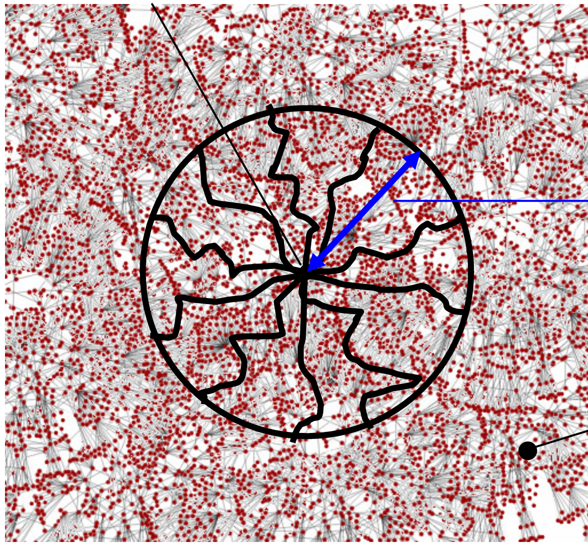


What if…?

ACOhg: Ant Paths Length

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

Initial node



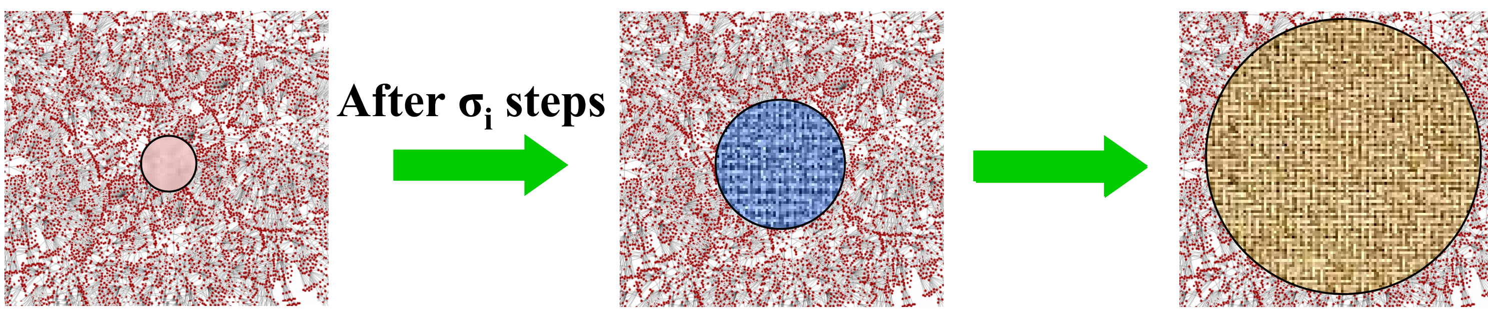
λ_{ant}

What if…?

Objective node

Two alternatives

Expansion Technique: λ_{ant} changes



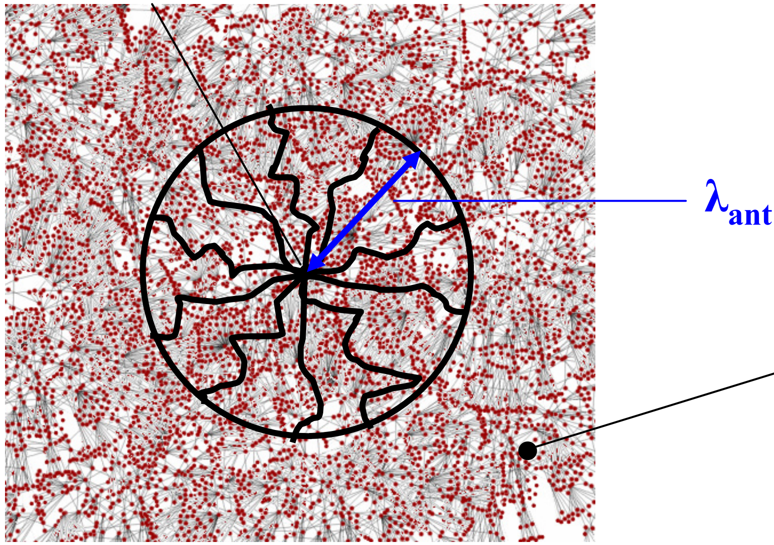
After σ_i steps

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

ACOhg: Ant Paths Length

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

Initial node

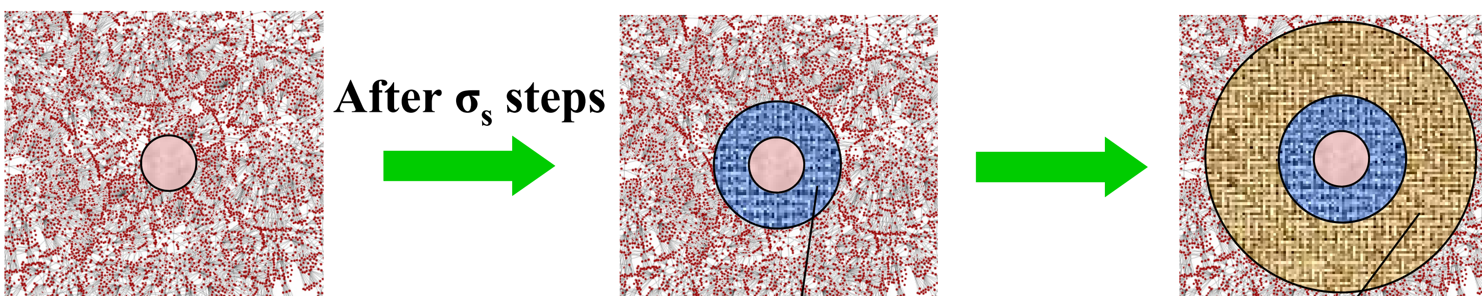


What if…?

Objective node

Two alternatives

Missionary Technique: starting nodes for path construction change



Second stage

Third stage

Introduction

ACOhg

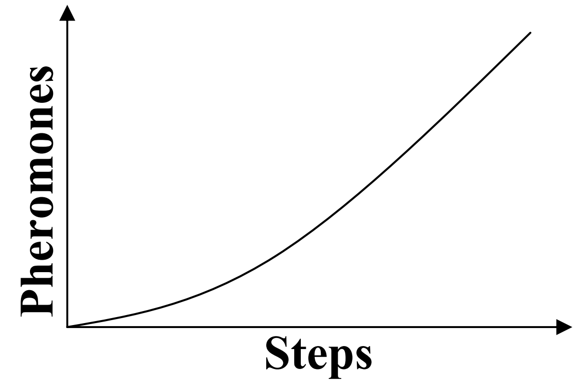
Systems Verification

Experiments

Conclusions & Future Work

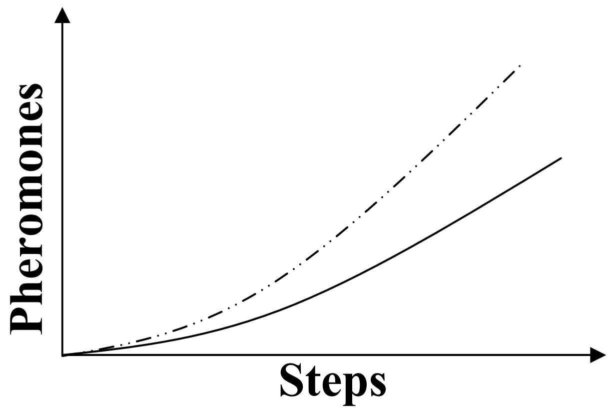
ACOhg: Pheromones

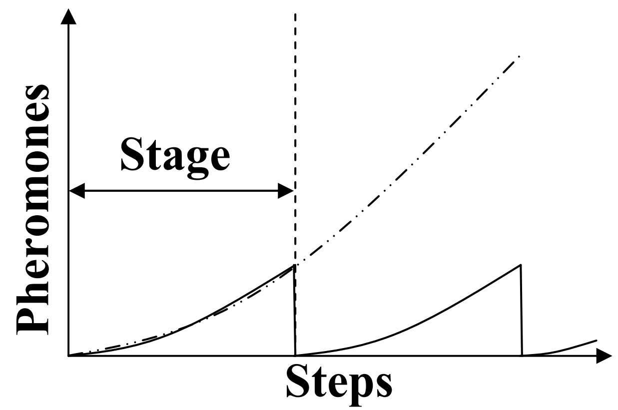
- The **number of pheromone** trails increases during the search
- This leads to memory problems
- We must **remove** some pheromone trails from memory



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

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

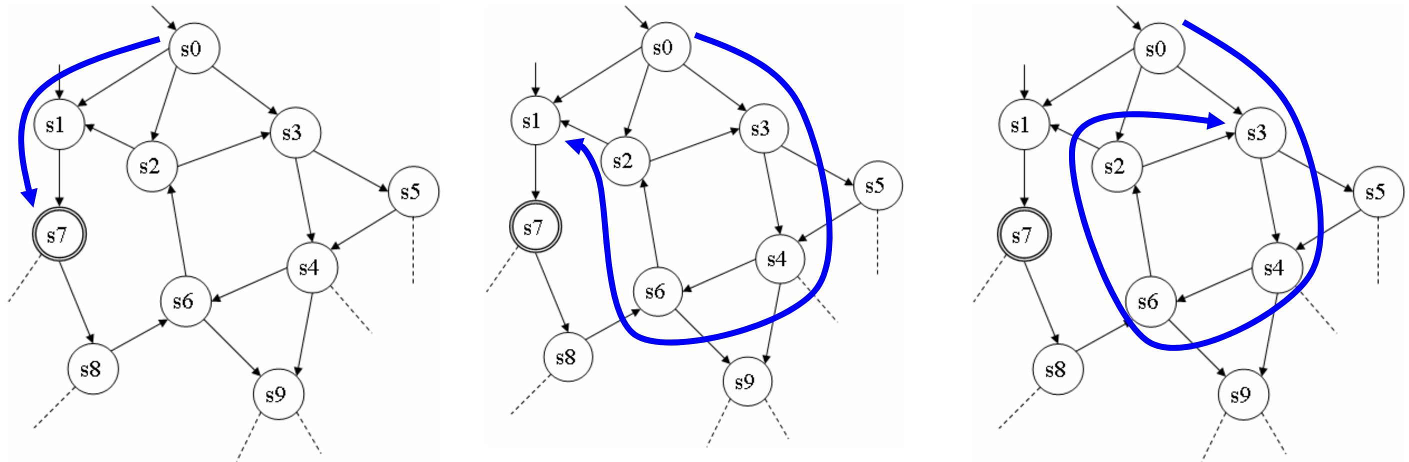




Introduction
ACOhg
Systems Verification
Experiments
Conclusions & Future Work

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	Partial solution without cycle	Partial solution with cycle
$p = 0$	$p = p_p$	$p = p_p + p_c \frac{\lambda_{ant} - l}{\lambda_{ant} - 1}$

Total penalty

Penalty constant for partial solutions

Penalty constant for solutions with cycles

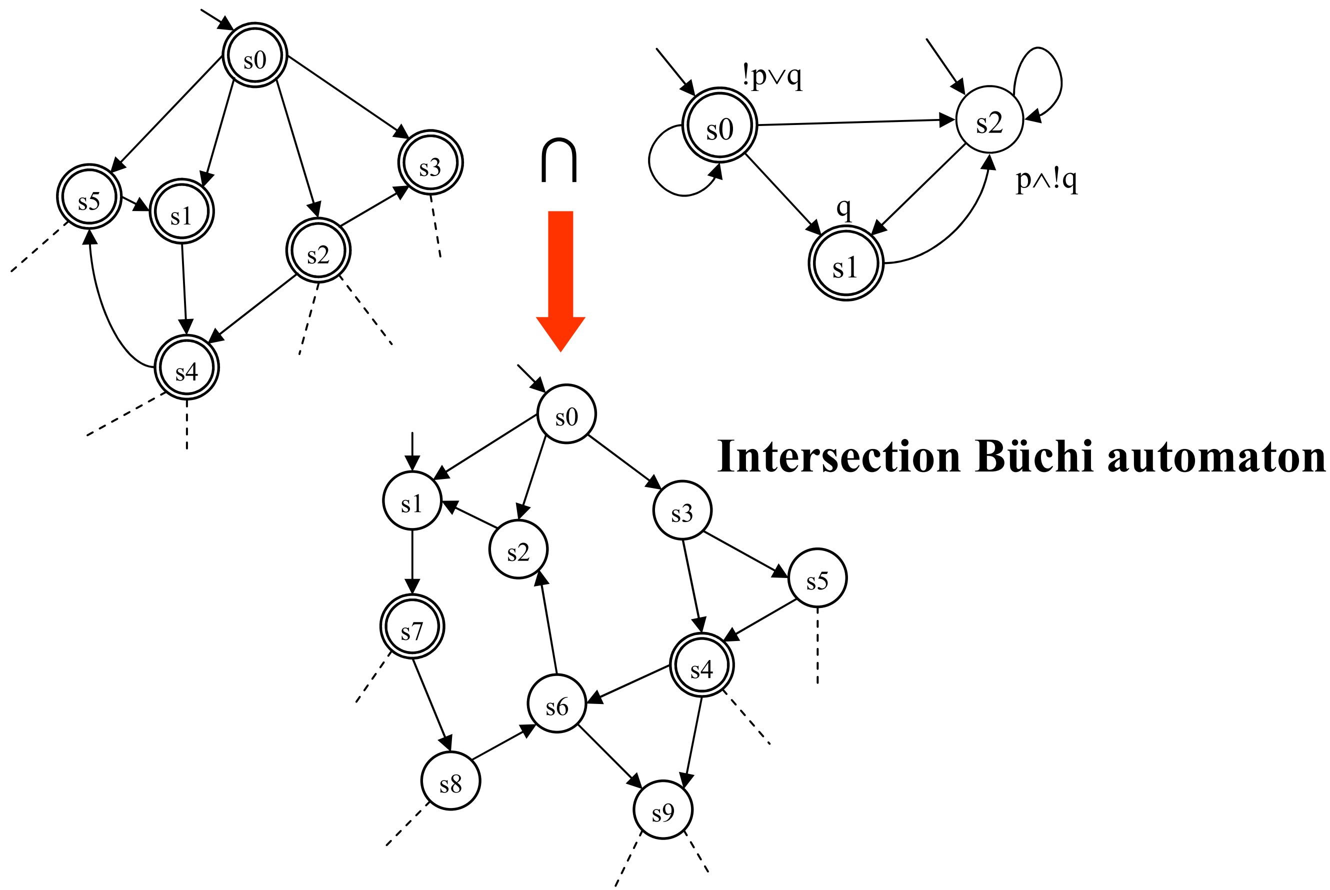
Path length

Systems Verification (I)

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

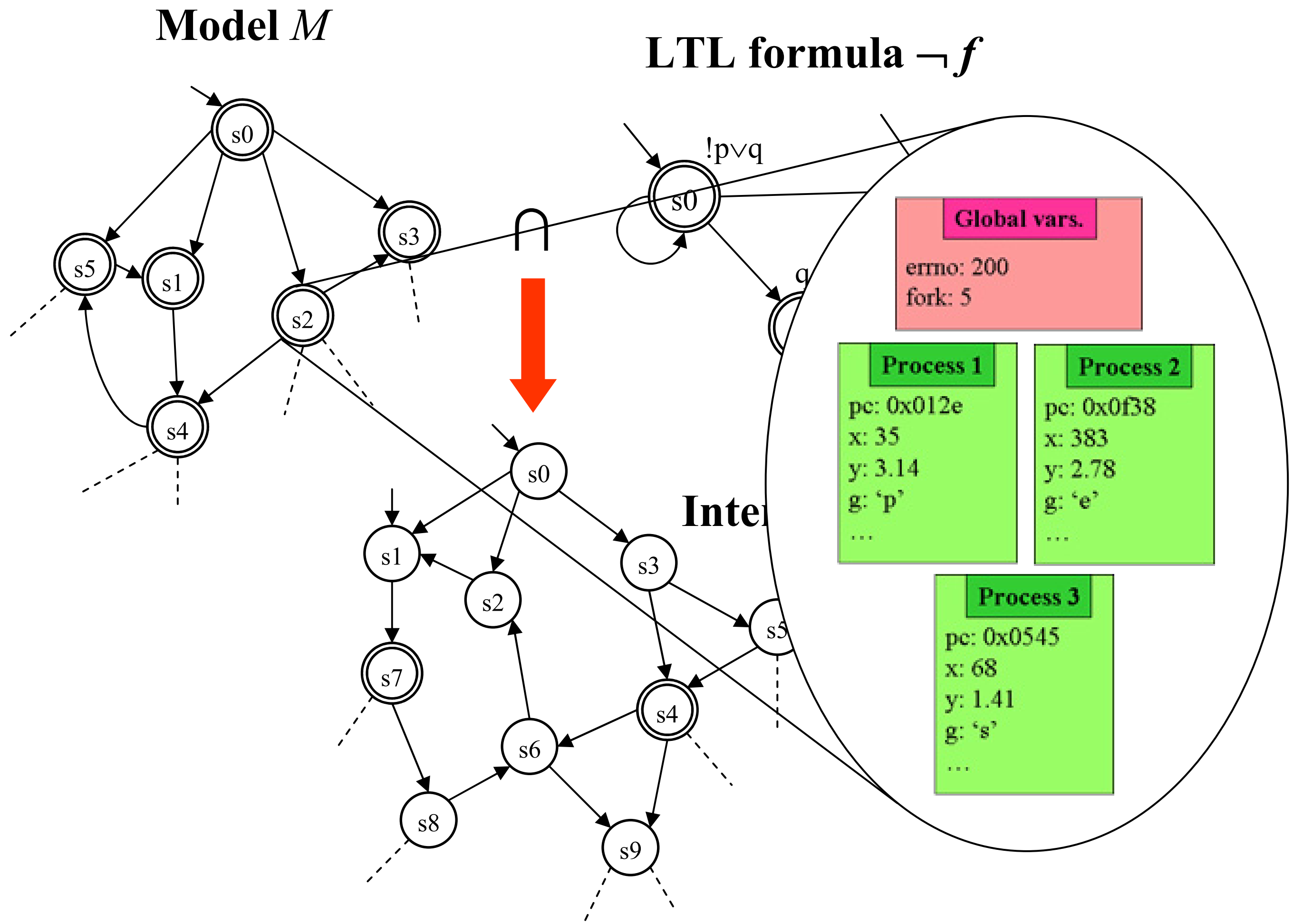
Model M

LTL formula $\neg f$



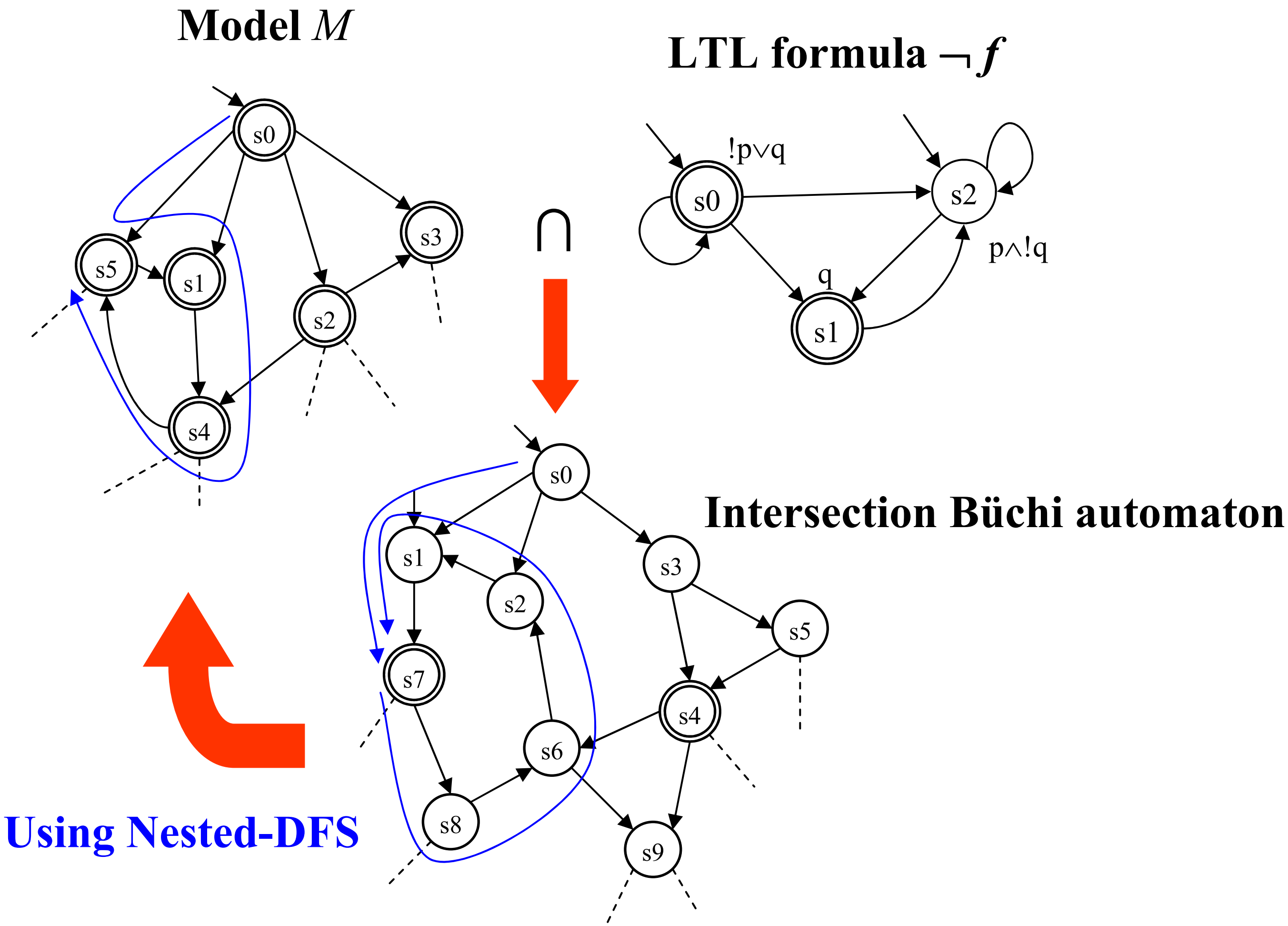
Systems Verification (I)

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



Systems Verification (I)

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



Systems Verification (II)

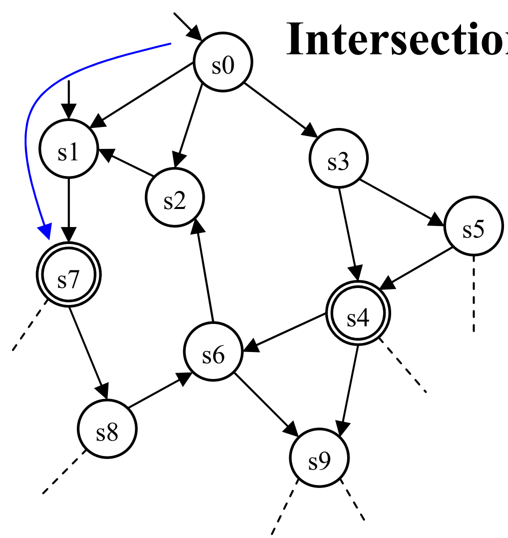
- **Safety properties are those expressed by an LTL formula of the form:**

$$f = \Box p$$

where *p* is a past formula (with only past operators)

- **Finding one counterexample ≡ finding one accepting state**

Intersection automaton

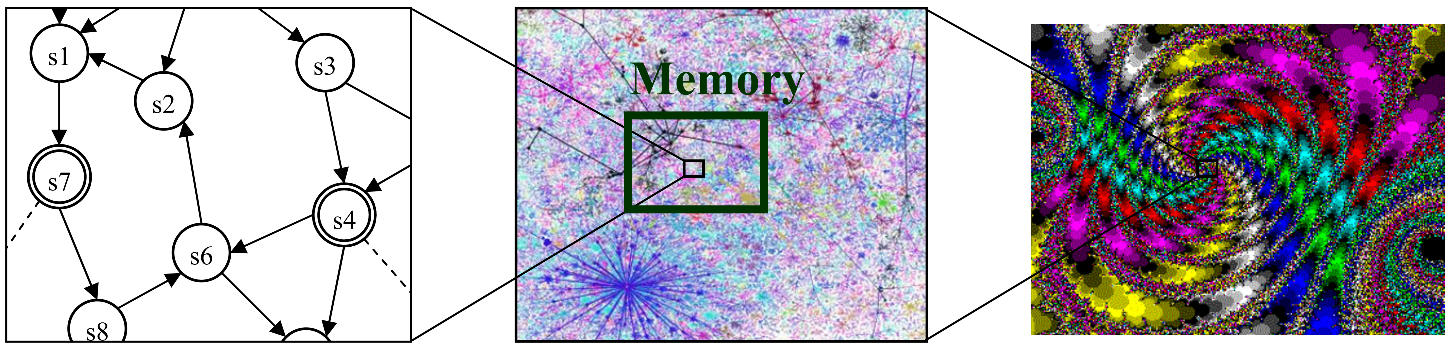


Safety Properties
Deadlocks
Invariants
Assertions
…

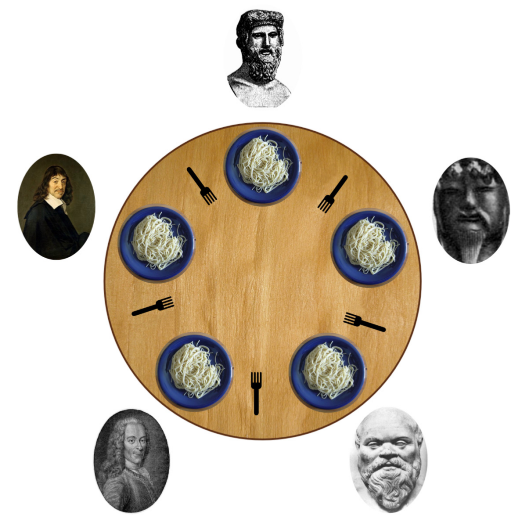
- **Classical algorithms for graph exploration can be used: DFS and BFS**

Systems Verification (and III)

- **Number of states very large even for small models**



- **The model used in the experiments implements the Dijkstra Dining Philosophers problem**
  - ➢ ***n* philosophers → 3^n states**
  - ➢ **1 deadlock state**
  - ➢ **20 philsophers → 1039 GB for storing the states**
  - ➢ **Memory required for storing all the pheromone trails: 520 GB**



Experiments: Parameters

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

Parameter	Value	Parameter	Value
Steps	10	a	5
Colony size	5	ρ	0.4
λ_{ant}	10	α	1.0
σ_s	2	β	1.0
s	10	p_p	1000
ξ	0.8	p_c	1000

- Fitness function: length of the path + heuristic + penalty
- Machine: Pentium 4 at 2.8 GHz with 512 MB
- Independent runs: 100

GECCO 2007

Experiments: Comparison

- We compare ACOhg against traditional exhaustive algorithms like DFS, BFS, and BF

	Hit (%)	Length	Mem. (KB)	Time (ms)
ACOhg	64	35.88	8467.06	271.56
DFS	0	-	-	-
BFS	0	-	-	-
BF	100	101.00	15360.00	60.00

- The only algorithms that are able to find errors in phi20 are BF and ACOhg
- The error trails found by ACOhg are shorter (better) than that obtained with BF
- ACOhg required half of the memory required by BF to find better error trails
- The hit rate of ACOhg can be increased changing the configuration

GECCO 2007

Introduction

ACOhg

Systems Verification

Experiments

Conclusions & Future Work

Experiments: Missionary (hit rate)

- We analyze the performance of the missionary technique

Hit rate (%)

σ_s	λ_{ant}				
	5	**10**	**15**	**20**	**25**
1	38	91	99	100	100
2	10	64	95	99	100
3	0	41	89	99	100
4	0	39	84	98	100
5	0	0	63	84	99
6	0	0	61	85	97
7	0	0	51	84	96
8	0	0	40	76	95
9	0	0	17	53	82
10	0	0	0	0	60

+ stages

- stages

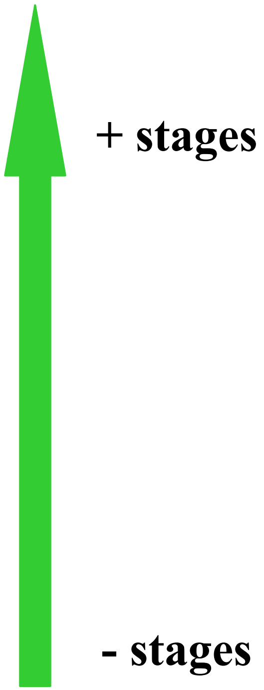
- The hit rate increases with λ_{ant} and with the number of stages
- The algorithm reaches deeper regions and the probability of finding the deadlock state is higher

Experiments: Missionary (length)

- We analyze the performance of the missionary technique

Error trail length (states)

σ_s	λ_{ant}				
	5	**10**	**15**	**20**	**25**
1	36.58	51.73	56.64	58.20	55.28
2	22.60	35.88	41.84	41.57	42.20
3	-	26.95	32.33	35.10	34.36
4	-	25.31	28.90	31.08	33.96
5	-	-	24.68	28.19	30.98
6	-	-	23.75	29.05	30.44
7	-	-	25.31	28.57	27.79
8	-	-	24.80	27.95	27.99
9	-	-	24.76	26.58	27.63
10	-	-	-	-	22.87



- The length of the error trails increases with λ_{ant} and with the number of stages
- The algorithm finds the deadlock state in deeper regions

Introduction

ACOhg

Systems Verification

Experiments

Conclusions & Future Work

Experiments: Missionary (memory)

- We analyze the performance of the missionary technique

Memory (KB)

σ_s	λ_{ant} = 5	10	15	20	25
1	4016.89	6436.65	8310.70	9719.76	10280.96
2	5507.40	8467.06	11210.11	13229.25	14399.85
3	-	10364.88	15118.38	18245.82	19191.94
4	-	13180.72	18834.29	23071.35	24465.29
5	-	-	22641.78	27928.38	28289.27
6	-	-	26523.28	32635.48	31614.64
7	-	-	30378.67	37388.19	33507.68
8	-	-	34124.80	42213.05	37339.66
9	-	-	37827.76	46930.11	40800.35
10	-	-	-	-	32902.08

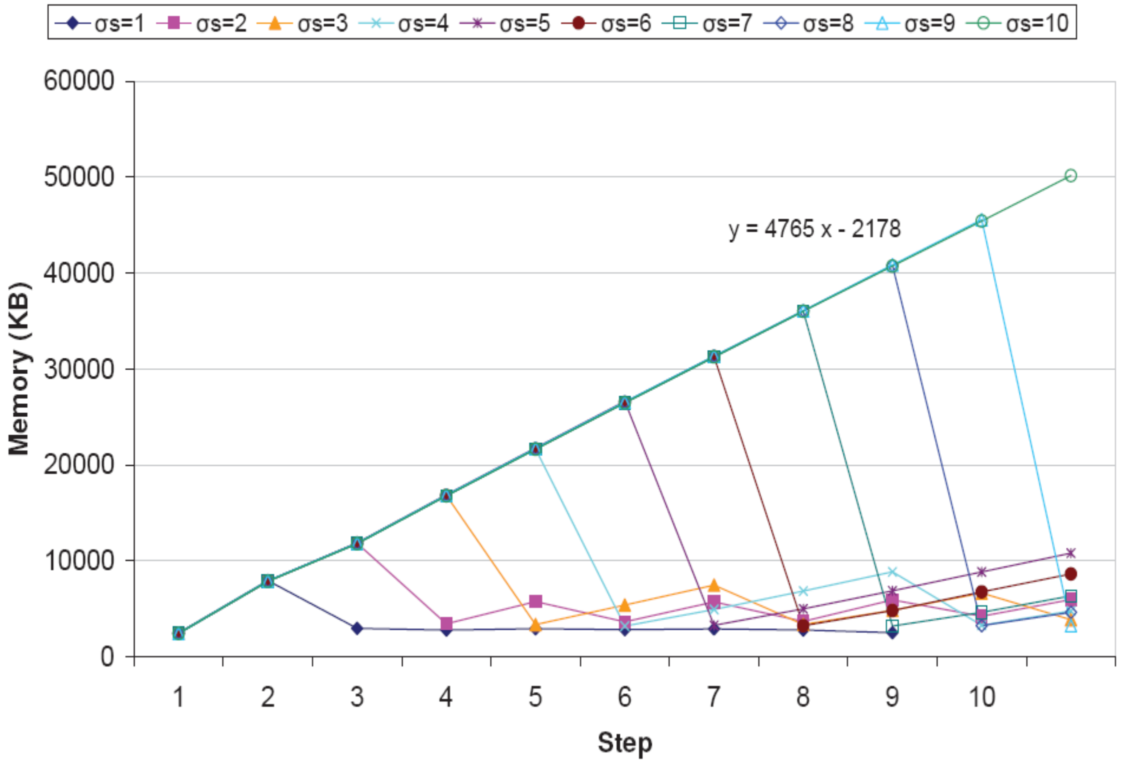
+ stages

- stages

- The required memory increases with λ_{ant} and decreases with the number of stages
- Traditional ACO with the same pheromone model: min. 1560 GB

Experiments: Missionary (memory)

- We analyze the performance of the missionary technique



$\lambda_{ant}=20$

- The pheromone reset after each stage keeps the memory consumption below an upper bound
- The slope of the line increases with λ_{ant}

Introduction

ACOhg

Systems Verification

Experiments

Conclusions & Future Work

Experiments: Missionary (time)

- We analyze the performance of the missionary technique

CPU time (milliseconds)

	λ_{ant}				
σ_s	**5**	**10**	**15**	**20**	**25**
1	96.05	147.03	176.46	207.60	221.40
2	168.00	271.56	321.16	382.42	395.30
3	-	381.95	470.79	580.20	565.50
4	-	520.51	653.81	810.20	820.40
5	-	-	837.14	1093.81	1023.54
6	-	-	1070.00	1406.47	1258.66
7	-	-	1317.45	1741.55	1411.46
8	-	-	1578.75	2135.39	1653.47
9	-	-	1872.35	2555.66	1967.20
10	-	-	-	-	953.67

+ stages

- stages

- As in the case of the memory, CPU time increases with λ_{ant} but decreases with the number of stages

- In any case, no more than 2.60 seconds are required

Experiments: Missionary (reset)

- **We analyze the performance of the missionary technique**

Pheromone reset

σ_s	No reset			Reset		
	Hit	Len.	Mem.(KB)	Hit	Len	Mem.(KB)
1	100	43.68	11519.17	100	41.52	10112.95
2	100	33.88	16555.47	100	35.72	14171.23
3	100	31.76	20146.70	100	33.56	19660.60
4	100	28.60	24442.83	100	30.00	23982.08
5	100	27.00	28595.77	99	29.85	27962.86
6	100	27.68	33315.26	100	28.88	30155.75
7	99	27.10	36525.46	100	28.20	35313.72
8	99	26.37	38855.26	97	27.23	40044.09
9	93	25.52	39883.61	95	26.64	40853.95
10	50	23.24	30753.10	56	23.07	31547.95

+ stages

- stages

- **The pheromone reset has a negligible influence on the hit rate and the error trail length (3 differences statistically significant)**
- **However, it reduces the required memory (7 differences stat. sign.)**

Introduction

ACOhg

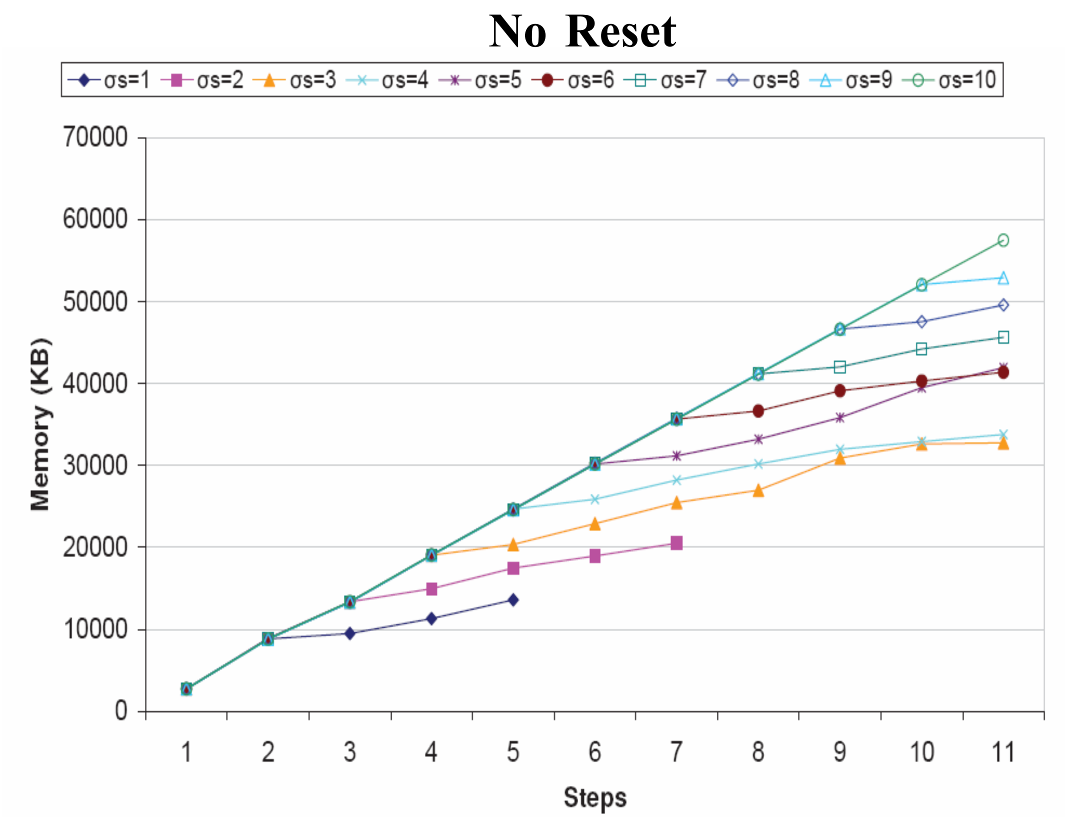
Systems Verification

Experiments

Conclusions & Future Work

Experiments: Missionary (reset)

- We analyze the performance of the missionary technique

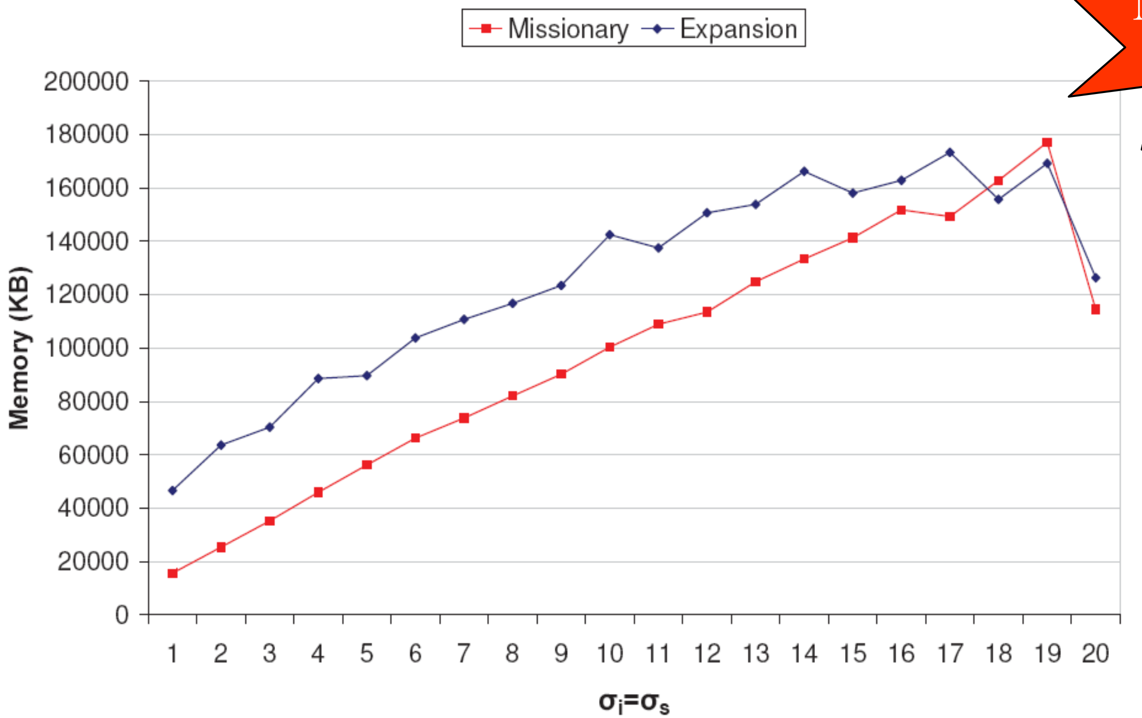


- Most of the memory is required in the first stage (in this problem)

Exps.: Expansion vs. Missionary

Not in the paper

- We compare expansion and missionary techniques



- Similar hit rate up to $\sigma_i=\sigma_s\approx 13$
- The average length of error trails is similar in both techniques
- Missionary technique requires less memory than expansion technique

Conclusions & Future Work

Conclusions

- ACOhg is able to overcome the limitations of current ACO models when dealing with problems with unknown or huge construction graphs
- ACOhg outperforms the traditional exhaustive algorithms from the model checking domain in the problem of finding safety errors in concurrent systems

Future Work

- Study the ACOhg model in depth (in progress)
- Transfer the ideas used in ACOhg to other metaheuristics in order to extend the set of problems to which they can be applied
- Design parallel versions of ACOhg able to be deployed in grid computing environments
- Study in depth the utilization of ACOhg for the problem of finding safety property violations in concurrent systems (Search-Based Software Engineering)

The End

Thanks for your attention !!!