An Ant Colony Optimisation Algorithm for the Set Packing Problem

Xavier GANDIBLEUX[1,2], Xavier DELORME[2] and Vincent T'KINDT[3]

(1) LINA - Laboratoire d'Informatique de Nantes Atlantique
Universite de Nantes
2 rue de la Houssiniere BP92208, F-44322 Nantes cedex 03 – FRANCE
Xavier.Gandibleux@lina.univ-nantes.fr

(2) LAMIH - Recherche Opérationnelle et Informatique
Université de Valenciennes et du Hainaut-Cambrésis
Le Mont Houy, F-59313 Valenciennes cedex 09 – FRANCE
Xavier.Delorme@univ-valenciennes.fr

(3) LI - Laboratoire d'Informatique
Polytech'Tours
64 avenue Jean Portalis, F-37200 Tours – FRANCE
Tkindt@univ-tours.fr

ANTS'04 – Fourth International Workshop on
Ant Colony Optimization and Swarm Intelligence
Bruxelles, Belgium, September 05–08, 2004

The Set Packing Problem (SPP)

Given

- a finite set $I = \{1, \ldots, n\}$ of items
- $\{T_j\}, j \in J = \{1, \ldots, m\}$, a collection of m subsets of I

a packing is a subset $P \subseteq I$ such that $|T_j \cap P| \leq 1, \forall j \in J$ which

$$\left[\begin{array}{ll} Max\ z(x) = & \sum_{i \in I} c_i x_i \\ & \sum_{i \in I} t_{i,j} x_i \leq 1, \forall j \in J \\ & x_i \in \{0,1\} \quad , \forall i \in I \\ & t_{i,j} \in \{0,1\} \quad , \forall i \in I, \forall j \in J \end{array}\right] \qquad (SPP)$$

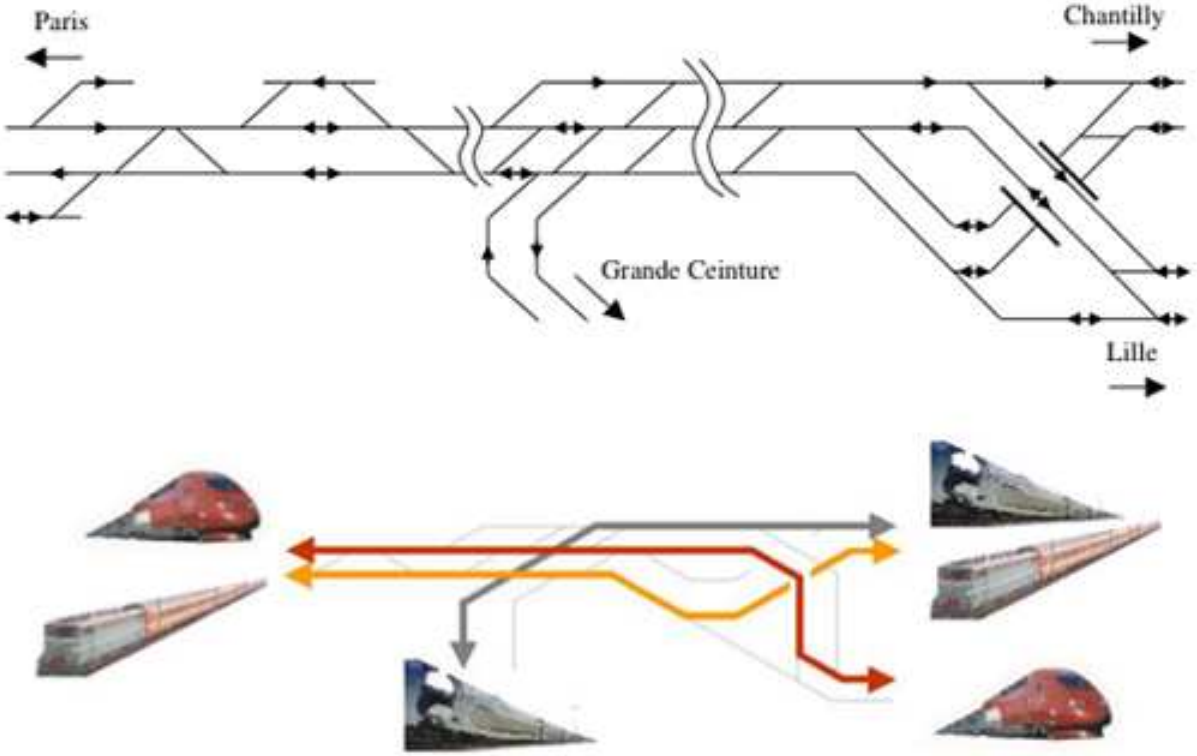
- Strongly NP-Hard (Garey and Johnson 1979)
- Node Packing Problem : $\sum_{i \in I} t_{i,j} = 2, \forall j \in J$

The literature about the SPP

- **Theoretical results and Exact algorithms** :
  - polyhedral theory, facets (Padberg 1973)
  - a Branch & Cut algorithm (Nemhauser 1999, Rossi 2001)
  - a lagrangian relax. with subgradient opt. (Rönnqvist 1995)
  - use a general solver (as LINDO) (Kim 1997)
- **Heuristics and meta-heuristics** :
  - GRASP with a path-relinking technique, a learning process and a dynamic tuning of parameters (Delorme 2004)
- **Practical applications** :
  - a cutting stock problem (Rönnqvist 1995)
  - a ship scheduling problem (Kim 1997)
  - bounds for a RCPSP (Mingozzi 1998)
  - a ground holding problem (Rossi 2001)
  - a railway planning problem (Zwaneveld 1996, Delorme 2003)

A railway planning problem

- Planning the construction or reconstruction of infrastructures
- Capacity of one component / junctions of a rail system
- Junction Pierrefitte-Gonesse, north of Paris



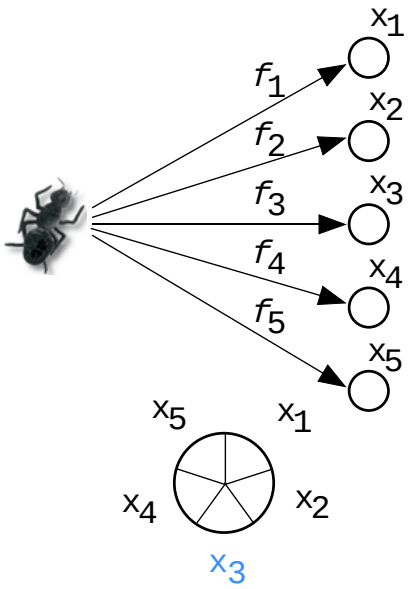
ACO principles for the SPP

A constructing solution process alternating *exploration mode* and *exploitation mode* with

- *more diversification at the beginning* of the solution process, and *more intensification at the end* (SACO, Stuztle 1998) :
  → probability evolves along the solution process
- application of a local search on each solution built by an ant (memetic algorithm, Murata 1995)

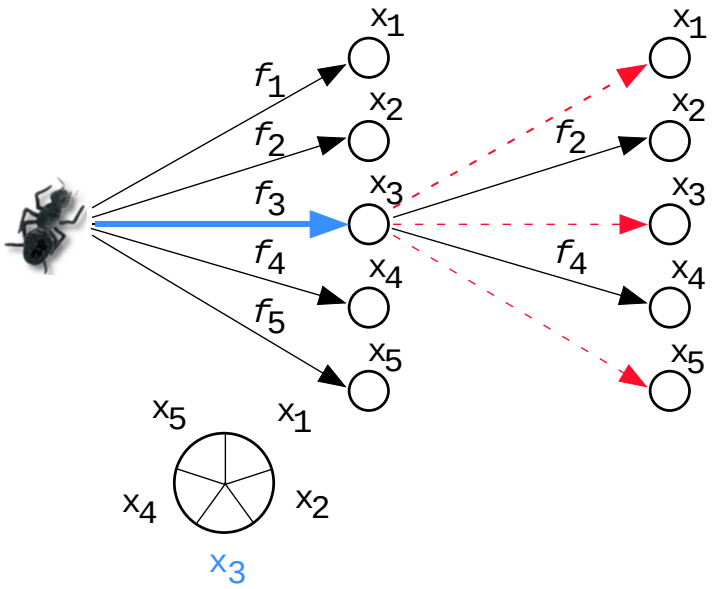
including **a territory disturance strategy** inspired by *a warming up strategy*, well-known for the SA

Construction of a solution (1/3)



x_1	x_2	x_3	x_4	x_5
.	.	.	.	.
1	0	1	0	1
1	1	0	0	0
0	1	0	0	1
0	0	0	1	0

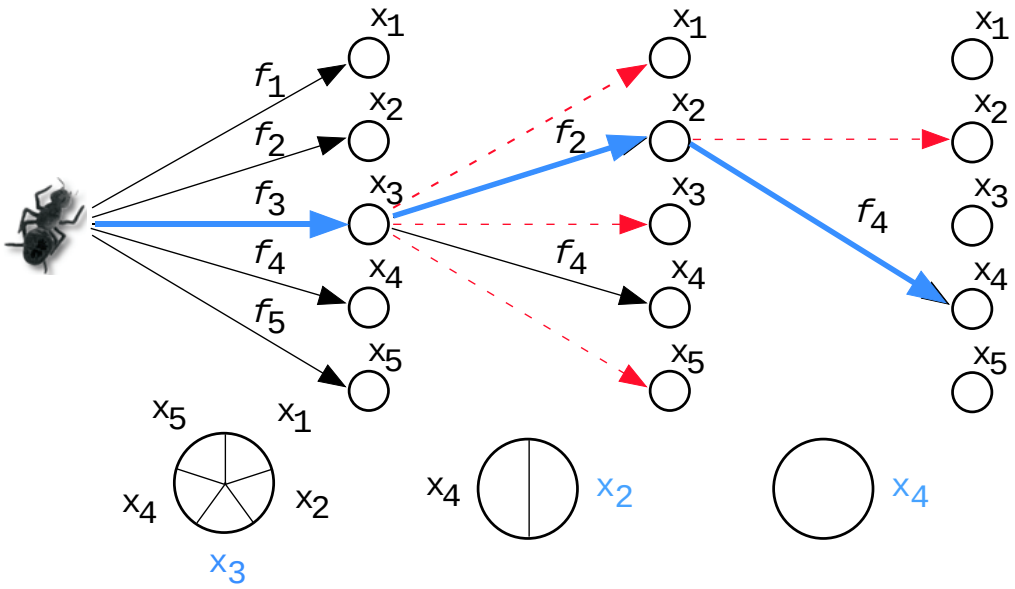
Construction of a solution (2/3)



x_1	x_2	x_3	x_4	x_5
.	.	.	.	.
1	0	1	0	1
1	1	0	0	0
0	1	0	0	1
0	0	0	1	0

x_1	x_2	x_3	x_4	x_5
0	.	1	.	0
~~1~~	0	1	0	~~1~~
1	1	0	0	0
0	1	0	0	1
0	0	0	1	0

Construction of a solution (3/3)



x_1	x_2	x_3	x_4	x_5
.	.	.	.	.
1	0	1	0	1
1	1	0	0	0
0	1	0	0	1
0	0	0	1	0

x_1	x_2	x_3	x_4	x_5
0	.	1	.	0
~~1~~	0	1	0	~~1~~
1	1	0	0	0
0	1	0	0	1
0	0	0	1	0

x_1	x_2	x_3	x_4	x_5
0	1	1	.	0
1	0	1	0	1
~~1~~	1	0	0	0
~~0~~	1	0	0	1
0	0	0	1	0

x_1	x_2	x_3	x_4	x_5
0	1	1	1	0
~~1~~	0	1	0	1
~~1~~	1	0	0	0
~~0~~	1	0	0	1
~~0~~	0	0	~~1~~	0

Outline of the algorithm

```
elaborateSolGreedy( sol ↑ ) ; localSearch( sol ↕ ) ; copySol( sol ↓ , bestSolKnown ↑)

initPheromones( φ ↑ ) ; iter ← 0
while not( isFinished?( iter ↓ ) ) do
  resetToZero( bestSolIter ↑ )
  for ant in 1... maxAnt do
    if isExploitation?(ant ↓, iter ↓, iterOnExploit ↓, maxIter ↓ ) then
      elaborateSolutionGreedyPhi( φ ↓ , solution ↑ )
    else
      elaborateSolutionSelectionMethod( φ ↓ , solution ↑ )
    end if
    localSearch( sol ↕ )
    if performance( sol ) > performance( bestSolIter ) then
      copySol( sol ↓ , bestSolIter ↑ )
      if performance( sol ) > performance( bestSolKnown ) then
        copySol( sol ↓ , bestSolKnown ↑ )
      end if
    end if
  end for
  managePheromones( φ ↕ , bestSolKnown ↓ , bestSolIter ↓ ) ;  iter++
end while
```

--| **The** `elaborateSolutionGreedyPhi` **procedure**

$I_t \leftarrow I \;\; ; \;\; x_i \leftarrow 0, \forall i \in I_t$

while $(I_t \neq \emptyset)$ **do**

$i^* \leftarrow \text{bestValue}(\phi_i, i \in I_t)$

$x_{i^*} \leftarrow 1 \;\; ; \;\; I_t \leftarrow I_t \setminus \{i^*\} \;\; ; \;\; I_t \leftarrow I_t \setminus \{i : \exists j \in J, t_{i,j} + t_{i^*,j} > 1\}$

end while

--| **The** `elaborateSolutionSelectionMode` **procedure**

$I_t \leftarrow I \;\; ; \;\; x_i \leftarrow 0, \forall i \in I_t$

$\mathcal{P} \leftarrow \log_{10}(\texttt{iter}) / \log_{10}(\texttt{maxIter})$

while $(I_t \neq \emptyset)$ **do**

if $(\text{randomValue}(0,1) > \mathcal{P})$ **then**

$i^* \leftarrow \text{rouletteWheel}(\phi_i, i \in I_t)$

else

$i^* \leftarrow \text{bestValue}(\phi_i, i \in I_t)$

end if

$x_{i^*} \leftarrow 1 \;\; ; \;\; I_t \leftarrow I_t \setminus \{i^*\} \;\; ; \;\; I_t \leftarrow I_t \setminus \{i : \exists j \in J, t_{i,j} + t_{i^*,j} > 1\}$

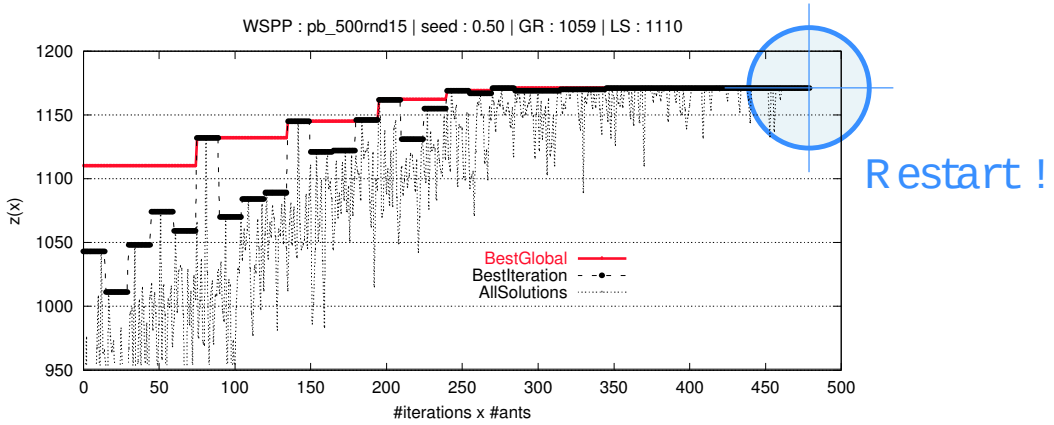
end while

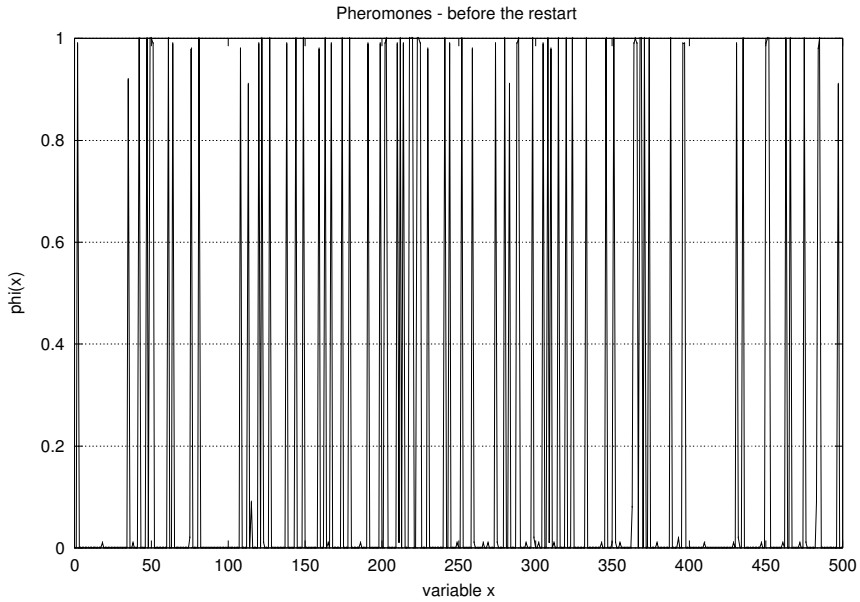
```
elaborateSolGreedy( sol ↑ ) ; localSearch( sol ↕ ) ; copySol( sol ↓ , bestSolKnown ↑)

initPheromones( φ ↑ ) ; iter ← 0
while not( isFinished?( iter ↓ ) ) do
  resetToZero( bestSolIter ↑ )
  for ant in 1... maxAnt do
    if isExploitation?(ant ↓, iter ↓, iterOnExploit ↓, maxIter ↓ ) then
      elaborateSolutionGreedyPhi( φ ↓ , solution ↑ )
    else
      elaborateSolutionSelectionMethod( φ ↓ , solution ↑ )
    end if
    localSearch( sol ↕ )
    if performance( sol ) > performance( bestSolIter ) then
      copySol( sol ↓ , bestSolIter ↑ )
      if performance( sol ) > performance( bestSolKnown ) then
        copySol( sol ↓ , bestSolKnown ↑ )
      end if
    end if
  end for
  managePheromones( φ ↕ , bestSolKnown ↓ , bestSolIter ↓ ) ;  iter ← iter + 1
end while
```

```
--| The managePheromones procedure
for i in 1 ... ncol do
   φ_i ← φ_i * rhoE                                   --| Pheromone evaporation
   if (bestSolIter.x[i] = 1) then
      φ_i ← φ_i + rhoD                                --| Pheromone deposition
   end if
end for
                                                      --| Territory disturbance
if      isStagnant?( bestSolution , iterStagnant )
   and isExistsPhiNul?( φ )
   and isRestartEnable?( iter , lastIterRestart , maxIteration ) then
   for i in 1... ncol do
      φ_i ← φ_i*0.95*log10(iter)/log10(maxIteration)   --| Disturb the pheromones
   end for
   for i in 1...random(0.0, 0.1 * ncol) do
      φ_random(1,ncol) ← random(0.05, (1.0 − iter/maxIteration) * 0.5)
   end for
   --| Offset on the pheromones with low level
   for i in 1... ncol do
      if φ_i < 0.1 then
         φ_i ← φ_i + random(0.05, (1.0 − iter/maxIteration) * 0.5)
      end if
   end for
end if
```

Illustration: pb500rnd15





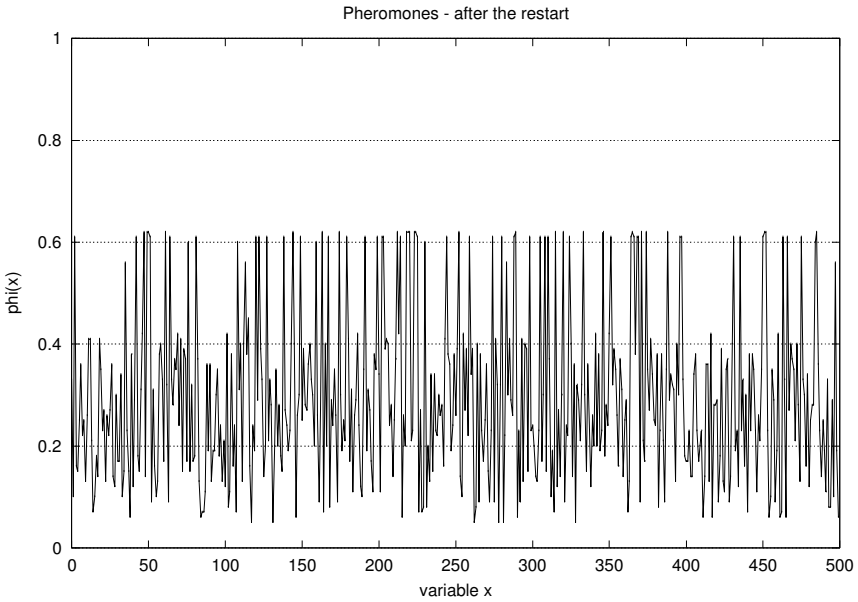
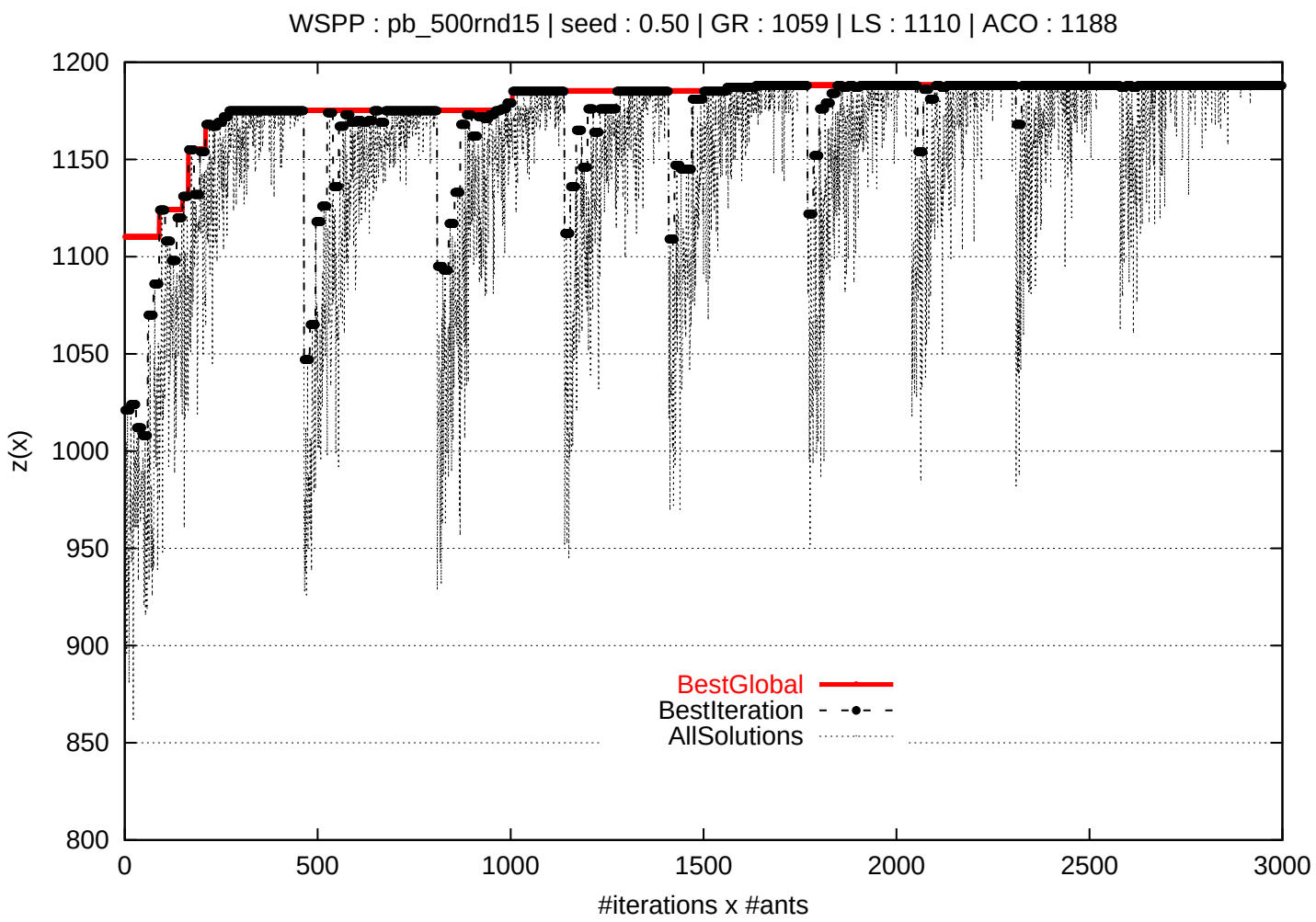
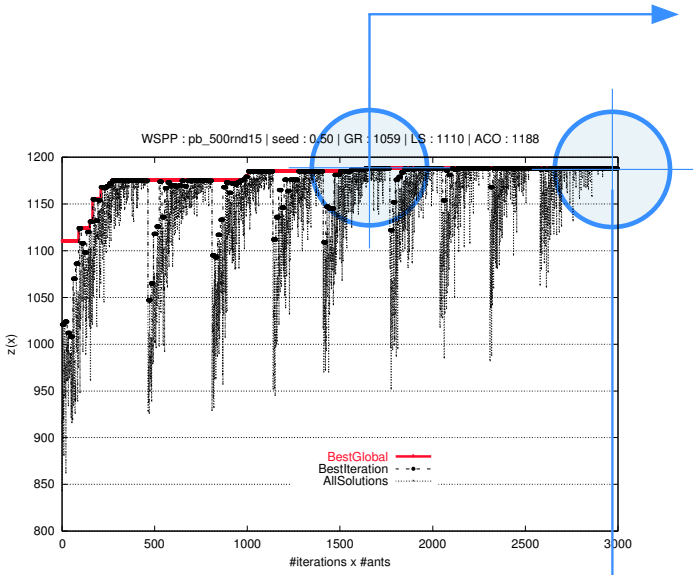
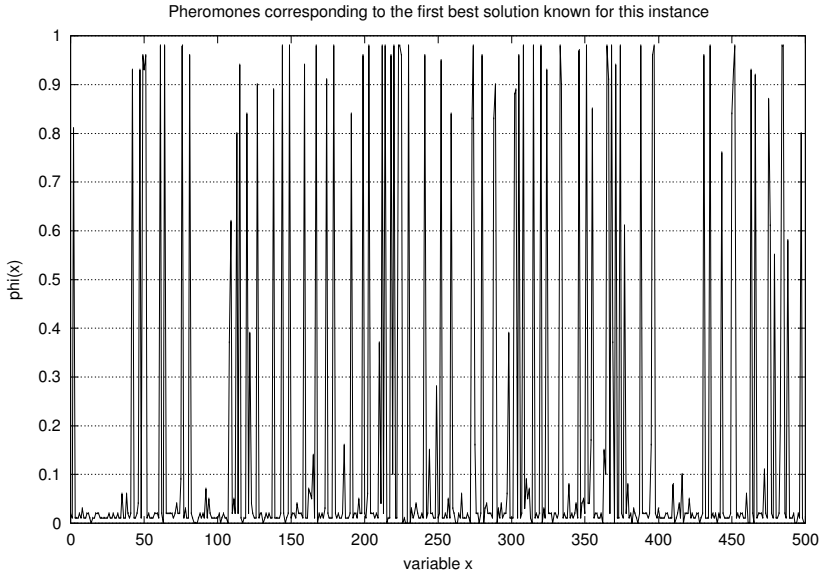
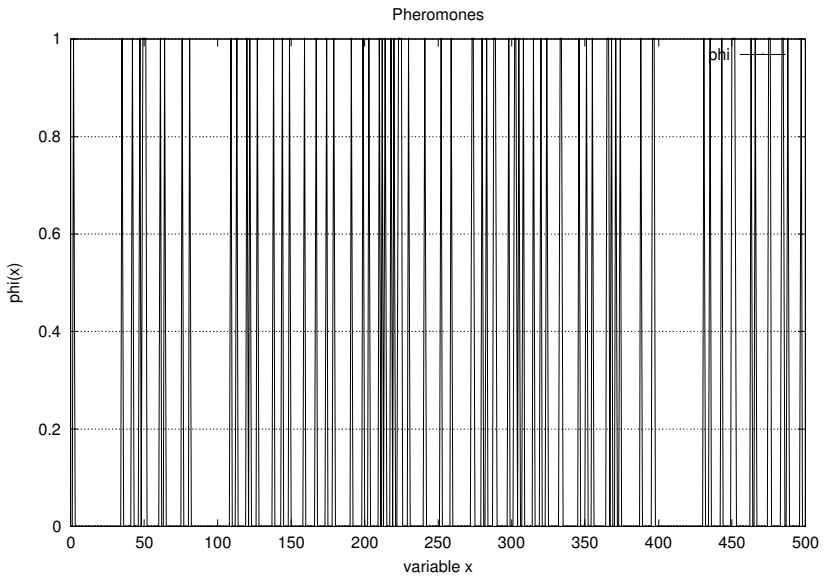


Illustration: $z(x)$ for pb500rnd15

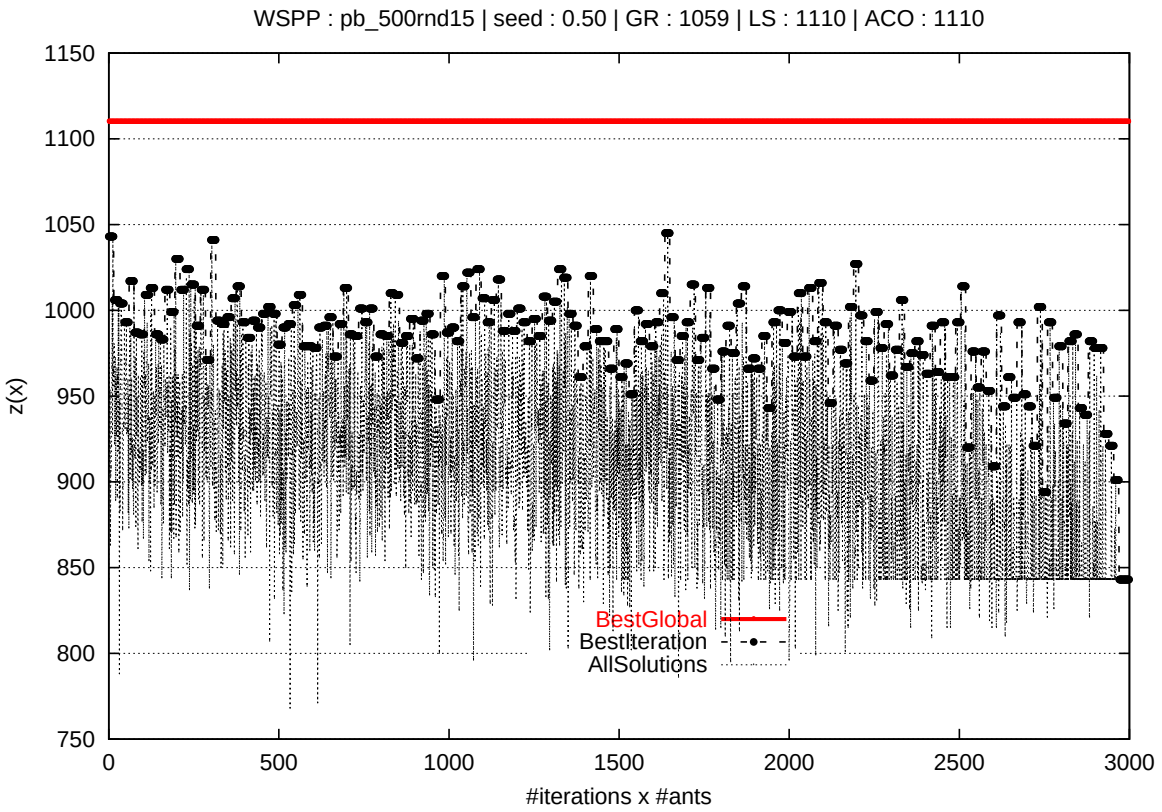



WSPP : pb_500rnd15 | seed : 0.50 | GR : 1059 | LS : 1110 | ACO : 1188
z(x)
#iterations x #ants
BestGlobal
BestIteration
AllSolutions


Pheromones corresponding to the first best solution known for this instance
phi(x)
variable x


Pheromones
phi
phi(x)
variable x

Contribution of the pheromone matrix



To see contribution of the pheromone matrix it would have been beneficial to note what was the performance of the ACO algorithm when alpha - weight for the pheromone information is set to zero → **Management of pheromone information disabled**

Numerical experiments

Parameters

`maxIter`	200	Number of iterations determined a priori
`maxAnt`	15	Number of ants for each iteration
`phiInit`	1.0	Initial pheromone assigned to a variable
`rhoE`	0.8	Pheromone evaporation rate
`rhoD`	phiInit * (1.0 - rhoE)	Pheromone deposition rate
`phiNul`	0.001	Level of pheromon considered as zero
`iterOnExploit`	0.750	Percentage of iterations when Exploitation mode is activated
`iterStagnant`	8	Declare the procedure stagnant when no improvement is observed

Experiments information

- Numerical instances
  - 100...1000 variables; 100...5000 constraints; WSPP/USPP
- Experimental environment
  - PC Pentium III; 800 MHz; 628 Mb; Debian
- Cplex
  - version 8.0.0
- GRASP (Delorme 2004)
  - Ada language; gnat 3.14
- ACO
  - C language; gcc 3.0.4 and O3

Data set 1: Instances100_200

100 variables:

- 100/300/500 constraints
- density of matrix between 2.0% and 3.1%
- USPP/WSPP [1,20]

200 variables:

- 200/600/1000 constraints
- density of matrix between 1.0% and 2.6%
- USPP/WSPP [1,20]

Inst.	0/1 Solution		GRASP		ACO	
	Opt.	CPUt(s)	Avg	CPUt(s)	Avg	CPUt(s)
100r01	372	2.92	372.00	1.97	372.00	3.33
100r02	34	0.60	34.00	1.31	34.00	2.00
100r03	203	7.81	203.00	1.14	203.00	2.00
100r04	16	52.86	16.00	1.29	15.56	0.67
100r05	639	0.01	639.00	0.80	639.00	1.67
100r06	64	0.01	64.00	0.69	64.00	1.00
100r07	503	0.00	503.00	1.00	503.00	1.00
100r08	39	0.02	38.75	0.57	38.68	0.67
100r09	463	0.49	463.00	1.26	463.00	1.67
100r10	40	1.13	40.00	1.28	39.62	1.00
100r11	306	0.48	306.00	0.68	306.00	1.67
100r12	23	6.80	23.00	1.13	22.93	0.33
200r01	416	8 760.73	415.18	7.32	415.25	27.33
200r02	32	156 109.36	32.00	7.35	31.56	14.67
200r03	731	5 403.23	722.81	10.81	725.12	44.33
200r04	64	63 970.91	63.00	9.12	62.93	24.33
200r05	184	1 211.37	184.00	4.62	182.56	16.00
200r06	14	8 068.20	13.37	3.48	12.87	4.00
200r07	1 004	0.02	1 001.12	4.20	1 003.50	6.33
200r08	83	0.04	82.87	2.71	82.75	2.67
200r09	1 324	0.01	1 324.00	3.75	1 324.00	7.33
200r10	118	0.02	118.00	3.64	118.00	4.00
200r11	545	0.33	544.75	2.36	545.00	4.33
200r12	43	1.70	43.00	1.01	43.00	1.33
200r13	571	830.39	566.43	6.01	568.50	20.33
200r14	45	10 066.91	45.00	3.92	44.43	8.67
200r15	926	12.20	926.00	4.22	926.00	27.00
200r16	79	14 372.85	78.31	6.80	78.37	15.33
200r17	255	741.52	251.31	3.61	253.25	11.00
200r18	19	19 285.06	18.06	2.35	18.12	3.00

Inst.	0/1 Solution		GRASP		ACO	
	Opt.	CPUt(s)	Avg	CPUt(s)	Avg	CPUt(s)
100r01	372	2.92	372.00	1.97	372.00	3.33
100r02	34	0.60	34.00	1.31	34.00	2.00
100r03	203	7.81	203.00	1.14	203.00	2.00
100r04	16	52.86	16.00	1.29	15.56	0.67
100r05	639	0.01	639.00	0.80	639.00	1.67
100r06	64	0.01	64.00	0.69	64.00	1.00
100r07	503	0.00	503.00	1.00	503.00	1.00
100r08	39	0.02	38.75	0.57	38.68	0.67
100r09	463	0.49	463.00	1.26	463.00	1.67
100r10	40	1.13	40.00	1.28	39.62	1.00
100r11	306	0.48	306.00	0.68	306.00	1.67
100r12	23	6.80	23.00	1.13	22.93	0.33
200r01	416	8 760.73	415.18	7.32	415.25	27.33
200r02	32	156 109.36	32.00	7.35	31.56	14.67
200r03	731	5 403.23	722.81	10.81	725.12	44.33
200r04	64	63 970.91	63.00	9.12	62.93	24.33
200r05	184	1 211.37	184.00	4.62	182.56	16.00
200r06	14	8 068.20	13.37	3.48	12.87	4.00
200r07	1 004	0.02	1 001.12	4.20	1 003.50	6.33
200r08	83	0.04	82.87	2.71	82.75	2.67
200r09	1 324	0.01	1 324.00	3.75	1 324.00	7.33
200r10	118	0.02	118.00	3.64	118.00	4.00
200r11	545	0.33	544.75	2.36	545.00	4.33
200r12	43	1.70	43.00	1.01	43.00	1.33
200r13	571	830.39	566.43	6.01	568.50	20.33
200r14	45	10 066.91	45.00	3.92	44.43	8.67
200r15	926	12.20	926.00	4.22	926.00	27.00
200r16	79	14 372.85	78.31	6.80	78.37	15.33
200r17	255	741.52	251.31	3.61	253.25	11.00
200r18	19	19 285.06	18.06	2.35	18.12	3.00

Data set 2: Instances500_1000

500 variables:

- 500/1500/2500 constraints
- density of matrix between 0.7% and 2.3%
- USPP/WSPP [1,20]

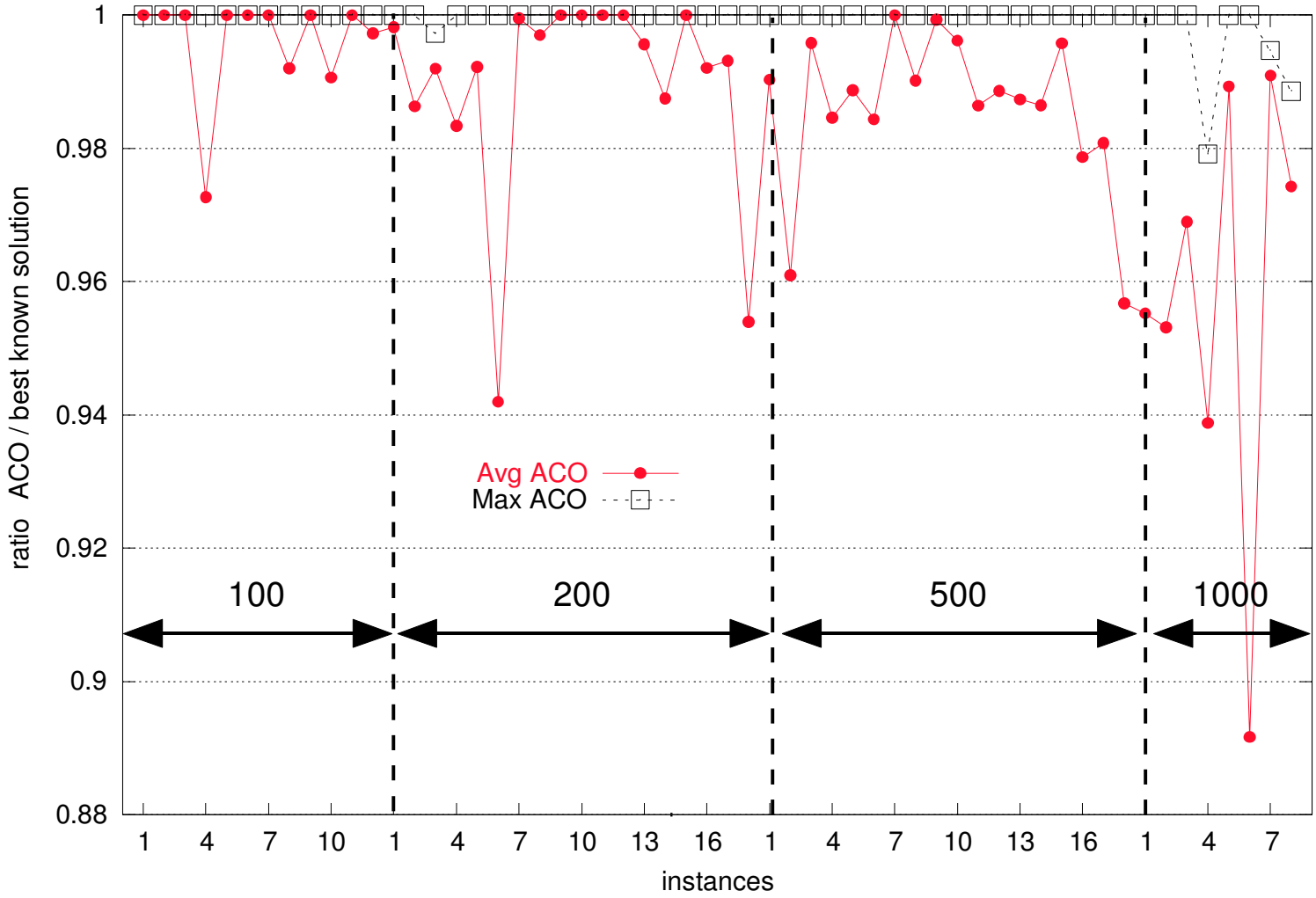
1000 variables:

- 1000/5000 constraints
- density of matrix between 0.58% and 2.65%
- USPP/WSPP [1,20]

Inst.	0/1 Solution	GRASP		ACO	
	Best known	Avg	CPUt (s)	Avg	CPUt (s)
500r01	323*	319.38	32.08	319.87	154.67
500r02	24*	23.69	25.62	23.06	26.67
500r03	776*	767.63	70.33	772.75	244.00
500r04	61*	60.13	57.30	60.06	69.00
500r05	122	121.50	15.48	120.62	71.00
500r06	8*	8.00	12.08	7.87	9.67
500r07	1 141	1 141.00	13.43	1 141.00	60.00
500r08	89	88.25	15.80	88.12	21.67
500r09	2 236	2 235.00	23.44	2 234.43	84.33
500r10	179	178.06	18.20	178.31	44.67
500r11	424	419.31	19.25	418.25	37.33
500r12	33*	33.00	11.91	32.62	8.00
500r13	474*	470.00	32.88	468.00	105.00
500r14	37*	36.94	20.77	36.50	25.66
500r15	1 196*	1 186.94	59.36	1 190.93	161.67
500r16	88*	86.63	36.31	86.12	60.67
500r17	192*	191.75	18.38	188.31	57.00
500r18	13*	13.00	12.03	12.43	9.00
1000r100	67	65.50	53.50	64.00	117.67
1000r200	4	3.15	39.30	3.81	15.00
1000r300	661*	639.50	221.20	640.50	700.67
1000r400	48*	46.83	149.70	45.06	108.33
1000r500	222*	217.98	64.80	219.62	85.67
1000r600	15*	13.68	41.40	13.37	8.00
1000r700	2 260	2 214.10	119.70	2 239.56	296.67
1000r800	175*	170.81	82.60	170.50	94.00

Inst.	0/1 Solution	GRASP		ACO	
	Best known	Avg	CPUt (s)	Avg	CPUt (s)
500r01	323*	319.38	32.08	319.87	154.67
500r02	24*	23.69	25.62	23.06	26.67
500r03	776*	767.63	70.33	772.75	244.00
500r04	61*	60.13	57.30	60.06	69.00
500r05	122	121.50	15.48	120.62	71.00
500r06	8*	8.00	12.08	7.87	9.67
500r07	1 141	1 141.00	13.43	1 141.00	60.00
500r08	89	88.25	15.80	88.12	21.67
500r09	2 236	2 235.00	23.44	2 234.43	84.33
500r10	179	178.06	18.20	178.31	44.67
500r11	424	419.31	19.25	418.25	37.33
500r12	33*	33.00	11.91	32.62	8.00
500r13	474*	470.00	32.88	468.00	105.00
500r14	37*	36.94	20.77	36.50	25.66
500r15	1 196*	1 186.94	59.36	1 190.93	161.67
500r16	88*	86.63	36.31	86.12	60.67
500r17	192*	191.75	18.38	188.31	57.00
500r18	13*	13.00	12.03	12.43	9.00
1000r100	67	65.50	53.50	64.00	117.67
1000r200	4	3.15	39.30	3.81	15.00
1000r300	661*	639.50	221.20	640.50	700.67
1000r400	48*	46.83	149.70	45.06	108.33
1000r500	222*	217.98	64.80	219.62	85.67
1000r600	15*	13.68	41.40	13.37	8.00
1000r700	2 260	2 214.10	119.70	2 239.56	296.67
1000r800	175*	170.81	82.60	170.50	94.00

ACO solution vs best known solution



Conclusion – Perspectives

- Even not sophisticated, ACO can find a good quality of solutions (including on large instances)
- ACO can find very good solutions, but it reacts differently according to the instances (convergence sometimes low?, more iterations?)
- A local search must be implemented for the USPP
- Faced with GRASP, this ACO version needs generally more CPUt, and sometimes takes a huge CPUt

$\Rightarrow$ ACO, in its current version, is a good starting point. A more efficient version can be expected in integrating more 'input' (coming from the ACO field or other metaheuristics) in the algorithm