Málaga, 2010

Informe Ejecutivo

TÍTULO: AFP-2.0: Un algoritmo de optimización basado en colonias de hormigas para el problema AFP

RESUMEN: En este entregable se detalla la implementación de un algoritmo basado en colonias de hormigas (*Ant Colony Optimization* o ACO) para la resolución del problema de la asignación automática de frecuencias. Se ha prestado especial atención en el diseño del algoritmo para que le permita resolver instancias de tamaño real. La efectividad de ACO se ha comparado con un algoritmo evolutivo básico para mostrar su efectividad.

OBJETIVOS:

1. Diseñar un algoritmo ACO para el problema AFP.
2. Uso de un método de mejora para el refinamiento de las soluciones.

CONCLUSIONES:

1. ACO ha mostrado ser un algoritmo muy competitivo para resolver instancias reales de AFP.
2. La utilización de algoritmos de búsqueda local es fundamental para abordar este problema con metaheurísticas.

RELACIÓN CON ENTREGABLES:

PRE: AFP-1.0, AFP-1.1

CO: —

Málaga, 2010

Executive Summary

TITLE: AFP-2.0: An ACO algorithm for the AFP problem

ABSTRACT: In this deliverable, the implementation of an Ant Colony Optimization Algorithm (ACO) for solving the AFP problem is detailed. Special attention has been paid to the design so that the algorithm allows addressing real-world instances. The effectiveness of ACO has been compared to a simple evolutionary algorithm.

GOALS:

1. Designing an ACO algorithm for the AFP problem.
2. Using local improvement operator for reaching more accurate solutions.

CONCLUSIONS:

1. ACO has shown to perform quite well for real-world AFP instances.
2. The usage of improvement operators is a must when solving AFP problems with metaheuristics.

RELATION WITH DELIVERABLES:

PRE: AFP-1.0, AFP-1.1

CO: —

An ACO algorithm for the AFP problem

DIRICOM

2010

1. Introduction

We have tackled the FAP with a standard ant colony optimization (ACO) [3] approach since this technique has provided good results when applied to other formulations of the FAP (see, for example, [8]). The algorithm has been evaluated on a real-world GSM network with 2612 transceivers which currently operates in a rather large U.S. city. After carefully tuning our algorithm, it has been compared against an evolutionary algorithm (EA) [7]. The ACO has shown to be very effective on the given problem instance, clearly outperforming the EA approach.

1.1. The ACO Approach

In general, our approach works as any other ACO algorithm: at each iteration candidate solutions are constructed in a probabilistic way. The probabilistic solution construction is based on a so-called pheromone model (denoted by $\mathcal{T}$), which is a set of numerical values that encode the algorithms' search experience. After the construction phase, some of the generated solutions are used to update the pheromone values in a way that aims at biasing the future solution construction towards good solutions found during the search process.

The particular approach that we implemented is known as $\mathcal{MM}$AS algorithm in the so-called hyper-cube framework (HCF); see [1]. A high level description of our algorithm is given in Algorithm 1. The data structures used by this algorithm, in addition to counters and to the pheromone model $\mathcal{T}$, are: (1) the *iteration-best* solution p_{ib}, that is, the best solution (frequency plan) generated in the current iteration by n_a ants; (2) the *best-so-far* solution p_{bs}, i.e., the best solution generated since the start of the algorithm; (3) the *restart-best* solution p_{rb}, that is, the best solution generated since the last restart of the algorithm; (4) the *convergence factor* cf, $0 \leq cf \leq 1$, which is a measure of how far the algorithm is from convergence; and (5) the Boolean variable *bs_update*, which becomes true when the algorithm reaches convergence.

Roughly, the algorithm works as follows. First, all the variables are initialized. In particular, the pheromone values are set to their initial value 0,5 by the procedure InitializePheromoneValues($\mathcal{T}$). Each algorithm iteration consists of the following steps. First, n_a solutions are generated by applying function GenerateFrequencyPlan($\mathcal{T}$). The iteration-best solution p_{ib} might be improved by the optional application of a local search method. Second, the values of the variables p_{ib}, p_{rb} and p_{bs} are updated in function Update(p_{ib},p_{rb},p_{bs}). Third, pheromone trail values are updated via the ApplyPheromoneUpdate(cf, *bs_update*, $\mathcal{T}$, p_{ib}, p_{rb}, p_{bs}) procedure. Fourth, a new value for the convergence factor cf is computed. Depending on this value, as well as on the value of the Boolean variable *bs_update*, a decision on whether to restart the algorithm or not is made. If the algorithm is restarted, then the procedure ResetPheromoneValues($\mathcal{T}$) is applied and all the pheromone values are reset to their initial value (set to 0.5). The algorithm is iterated until some opportunely defined termination conditions are satisfied. Once terminated, the algorithm returns the best-so-far solution s_{bs}. In the following we describe the main procedures of Algorithm 1 in more detail.

GenerateFrequencyPlan($\mathcal{T}$): ACO algorithms utilize a so-called pheromone model $\mathcal{T}$, which is a set of numerical values, for the probabilistic construction of solutions. Here, $\mathcal{T}$ consists of a pheromone value $\tau_{ij} > 0$ for each combination of a transceiver t_i and a valid frequency $f_{ij} \in F_i$. τ_{ij} represents the desirability of assigning frequency f_{ij} to transceiver t_i. A solution is constructed by choosing for each transceiver t_i a frequency from F_i in the order $i = 1, \ldots, n$. This is done as follows. First, at each construction step the following probabilities are generated:

$$\mathbf{p}(f_{ij}) = \frac{\tau_{ij} \cdot \eta_{ij}}{\sum_{f_{il} \in F_i} \tau_{il} \cdot \eta_{il}} \ , \ \forall \ f_{ij} \in F_i \tag{1}$$

The η values in this equation are called the *heuristic information*. They are generally defined by using the weights of a greedy heuristic for the tackled problem. Here we consider 3 different options:

- **Option 1:** No heuristic information, that is, all η_{ij} values are set to 1.
- **Option 2:** The objective function punishes the assignment of *too similar* frequencies to transceivers installed in the same sector. Therefore, we try to avoid using too similar frequencies as follows. Let T_i be the set of

Algorithm 1 Pseudo-code of the ACO approach

```
input: a problem instance
p_bs ← NULL, p_rb ← NULL, cf ← 0, bs_update ← false
InitializePheromoneValues(T)
while termination conditions not satisfied do
  for j ← 1 to n_a do
    p_j ← GenerateFrequencyPlan(T)
  end for
  p_ib ← argmin(C(p_1), ..., C(p_{n_a}))
  p_ib ← LocalSearch(p_ib,d)        /* optional */
  Update(p_ib,p_rb,p_bs)
  ApplyPheromoneUpdate(cf,bs_update,T,p_ib,p_rb,p_bs)
  cf ← ComputeConvergenceFactor(T)
  if cf > 0,99 then
    if bs_update = true then
      ResetPheromoneValues(T)
      p_rb ← NULL
      bs_update ← false
    else
      bs_update ← true
    end if
  end if
end while
output: p_bs
```

transceivers that are installed in sector $s(t_i)$, and let $\hat{T}_i \subset T_i$ be the set of transceivers that have already assigned a frequency. Then,

$$\eta_{ij} \leftarrow \left(\left(100 \cdot \sum_{t_l \in \hat{T}_i} \delta(p(t_l), f_{ij}) \right) + 1 \right)^{-1} , \tag{2}$$

where $\delta(x, y) = 1$ if $|x - y| \leq 1$, and $\delta(x, y) = 0$ otherwise. In this setting, we multiply the sum of the *interferences* with 100 in order to increase their importance with respect to the pheromone information. The value of 100 was decided by tuning by hand.

- **Option 3:** As the objective function is additive, we can compute the exact influence of assigning a frequency $f_{ij} \in F_i$ to a transceiver t_i with respect to the current partial solution. Let $\hat{T} = \{t_1, \ldots, t_{i-1}\}$ be the set of transceivers to which a frequency is already assigned. The increase of the objective function value caused by the setting $p(t_i) = f_{ij}$ can be computed as follows:

$$\Delta(p, p(t_i) = f_{ij}) = \sum_{t \in \hat{T}} (C_{\text{sig}}(p, t, t_i) + C_{\text{sig}}(p, t_i, t)) \tag{3}$$

Accordingly, we define

$$\eta_{ij} = ((100 \cdot \Delta(p, p(t_i) = f_{ij})) + 1)^{-1} \quad . \tag{4}$$

In order to choose one of the possible frequencies for transceiver t_i, we proceed as follows. First, a random number $r \in [0, 1]$ is generated. In case $r < d_{\text{det}}$, frequency $f_{ij} \in F_i$ is chosen such that $\mathbf{p}(f_{ij}) \geq \mathbf{p}(f_{il})$, $\forall f_{il} \in F_i$. Otherwise, a frequency for t_i is chosen probabilistically by roulette-wheel-selection with respect to the probabilities defined in Equation 1. Note that d_{det} is an important parameter of the algorithm. It is used to determine the percentage of deterministic versus probabilistic construction steps.

LocalSearch(p_{ib},d): Optionally we apply a simple local search method to the iteration-best solution p_{ib}. Parameter d indicates for how many steps this algorithm should be maximally executed (see Algorithm 2). This parameter is crucial due to the considerable computation time consumption of each local search step.

ApplyPheromoneUpdate(cf,bs_update,$\mathcal{T}$,p_{ib},p_{rb},p_{bs}): In general, three solutions are used for updating the pheromone values. These are the iteration-best solution p_{ib}, the restart-best solution p_{rb}, and the best-so-far solution p_{bs}. The influence of each solution on the pheromone update depends on the state of convergence of the algorithm as measured by the convergence factor cf. At the beginning of each restart phase (i.e., when bs_update = FALSE and cf is close to zero), the iteration-best solution p_{ib} has maximal influence. Then, as the algorithm progresses (i.e., when cf increases), the influence of the restart-best solution p_{rb} increases. Shortly before convergence (i.e., $cf \leq 0{,}99$, but close to 0,99), the restart-best solution p_{rb} has maximal influence, and once convergence is detected (i.e., $cf > 0{,}99$),

Algorithm 2 Pseudo-code for the local search

```
input: a solution p, a number of steps d
improved ← true
k ← 1
while k ≤ d and improved = true do
    improved ← false
    for i ← 1 to n do
        Replace frequency p(t_i) with the frequency from F_i that most reduces the objective function value
        if the objective function value was reduced then improved = true
    end for
    k ← k + 1
end while
output: a possibly improved solution p
```

Table 1: Setting of κ_{ib}, κ_{rb}, κ_{bs}, and ρ depending on the convergence factor cf and the Boolean control variable *bs_update*.

	bs_update = FALSE				*bs_update*
	$cf < 0{,}4$	$cf \in [0{,}4, 0{,}6)$	$cf \in [0{,}6, 0{,}8)$	$cf \geq 0{,}8$	= TRUE
κ_{ib}	1	2/3	1/3	0	0
κ_{rb}	0	1/3	2/3	1	0
κ_{bs}	0	0	0	0	1
ρ	0,2	0,2	0,2	0,15	0,15

the control variable *bs_update* is set to TRUE which has the effect that the best-so-far solution p_{bs} has maximal influence. This is done in order to intensify the search around the best-so-far solution and can have the effect to move the search to a different zone of the search space, with the consequent decrease of the convergence factor.

Each pheromone value $\tau_{ij} \in \mathcal{T}$ is updated as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \rho \cdot (\mu_{ij} - \tau_{ij}) \ , \tag{5}$$

where

$$\mu_{ij} \leftarrow \kappa_{ib} \cdot \delta(p_{ib}, f_{ij}) + \kappa_{rb} \cdot \delta(p_{rb}, f_{ij}) + \kappa_{bs} \cdot \delta(p_{bs}, f_{ij}) \ , \tag{6}$$

where κ_{ib} is the weight (that is, the influence) of solution p_{ib}, κ_{rb} is the weight of solution p_{rb}, κ_{bs} is the weight of solution p_{bs}, and $\kappa_{ib} + \kappa_{rb} + \kappa_{bs} = 1$. Moreover, $\delta(p, f_{ij}) = 1$ if $p(t_i) = f_{ij}$, and $\delta(p, f_{ij}) = 0$ otherwise. After the pheromone update rule (Equation 5) is applied, pheromone values that exceed $\tau_{\text{máx}} = 0{,}999$ are set back to $\tau_{\text{máx}}$ (similarly for $\tau_{\text{mín}} = 0{,}001$).

Equation 6 allows to choose how to schedule the relative influence of the three solutions used for updating the pheromone values. For our application we used the update schedule as shown in Table 1.

ComputeConvergenceFactor($\mathcal{T}$): The convergence factor cf, which is a function of the current pheromone values, is computed as follows:

$$cf \leftarrow 2 \left(\left(\frac{\sum_{\tau_{ij} \in \mathcal{T}} \text{máx}\{\tau_{\text{máx}} - \tau_{ij}, \tau_{ij} - \tau_{\text{mín}}\}}{|\mathcal{T}| \cdot (\tau_{\text{máx}} - \tau_{\text{mín}})} \right) - 0{,}5 \right)$$

In this way, $cf = 0$ when the algorithm is initialized (or reset), that is, when all pheromone values are set to 0,5. On the other side, when the algorithm has converged, then $cf = 1$. In all other cases, cf has a value in $(0, 1)$. This completes the description of the ACO algorithm.

2. Experimental evaluation

We implemented our algorithms in ANSI C++ using GCC 3.2.2 for compiling the software. Our experimental results were obtained on a PC with an AMD64X2 4400 processor and 4 Gb of memory, running Debian Linux.

2.1. Problem Instance

Here, we want to provide the reader with details on the FAP instance which is being tackled. The GSM network used has 711 sectors with 2,612 TRXs to be assigned a frequency; the constants in Equations 2, 3, and 6 were set to

Table 2: Numerical results concerning the heuristic information: Options 1, 2, and 3

	No local search			Local search		
	best	**average**	**std.**	**best**	**average**	**std.**
Option 1	67460090.28	69754591.73	1592508.81	90515.57	92021.80	983.03
Option 2	240193.31	242167.29	990.92	89703.44	91064.36	887.85
Option 3	154835.98	158307.52	2140.39	91496.06	93494.52	1511.98

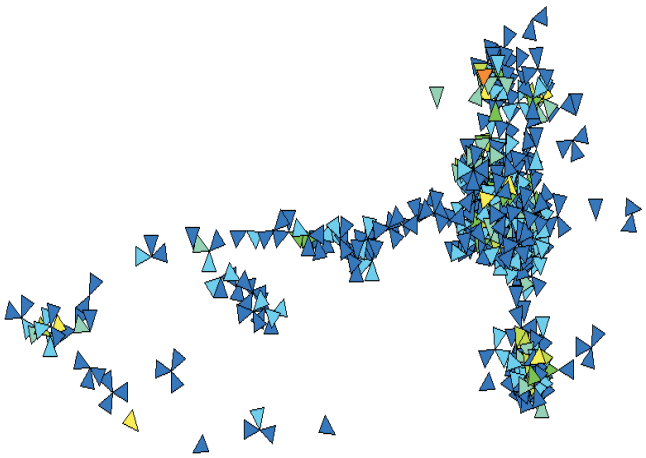

Figura 1: Topology of the GSM instance used.

$K = 100,000$, $c_{\text{SH}} = 6\ dB$, and $c_{\text{ACR}} = 18\ dB$, respectively. Each TRX has 18 available channels (from 134 to 151). Figure 1 displays the network topology, every triangle representing a sectorized antenna in which operate several TRXs.

This GSM network is currently operating in a U.S. 400 km^2 city with more than 500,000 people, so its solution is of great practical interest. The data source to build the interference matrix based on the C/I probability distribution uses thousands of Mobile Measurement Reports (MMRs) [6] rather than propagation prediction models. MMRs are a more accurate data source, as they capture the call location pattern in the network and do not rely on predictions. These properties make our GSM problem more realistic than standard available benchmarks [5]. Indeed, the most similar available instances are the COST 259 benchmark, but the basic traffic load is drawn at random according to an empirically observed distribution, and signals are predicted with several propagation models. The Philadelphia, CELAR and GRAPH instances [5] are even simpler. Moreover, the largest instance in [5] scales only up to 1886 cells. Other realistic-sized GSM instances exist in the literature, e.g. [2], but neither they are publicly available nor they use accurate interference information as our mathematical model requires.

2.2. Tuning of the ACO algorithm

In the following we present the results of tuning experiments that were performed in order to fix the parameters of the ACO algorithm. The first set of experiments concerns the three options for the heuristic information (see Section 1.1). We applied the ACO algorithm with the following fixed settings: $n_a = 5$, that is, 5 ants per iteration,[1] and $r_{\text{det}} = 0{,}5$. The local search algorithm was applied with $d = 3$, that is, maximally 3 steps per application. We applied the ACO algorithm with all three options for the heuristic information (with and without local search) to the given problem instance. For all the runs we used a time limit of 600 seconds. The results—averaged over 10 independent runs—are shown in Table 2. For each setting (for example, Option 1 without local search) we provide the best value found in 10 runs (column **best**), the average over 10 runs (column **average**), and the standard deviation (column **std.**). The results allow us to draw the following conclusions. When no local search is used, the heuristic information plays a crucial role. In fact, the algorithm with Option 2, respectively Option 3, is around 280 times, respectively 430 times, better than the algorithm version using Option 1 (that is, no heuristic information). When comparing between using Options 2 and 3, we note a clear advantage of the algorithm version using Option 3. Once local search is used, the three algorithm versions do not differ that much. In fact, the best algorithm version is now the one using Option 2. The reason is based in the computation time needed for constructing a solution. While the computations required by Option 2 do not cause a significant increase in the computation time needed to construct a solution, the use of Option 3 is quite expensive. More in detail, the construction of a solution with Option 3 is

[1] Note that this setting is not crucial. Hence, we just chose a reasonable value without performing tuning experiments.

Table 3: Numerical results of ACO, ACO*, and EA for 3 different time limits

	ACO			ACO*			EA			
time limit	**best**	**average**	**std.**	**best**	**average**	**std.**	**best**	**average**	**std.**	
120 s	91140.04	93978.21	1165.93	104719.72	106330.66	742.47	104247.72	108071.98	1723.40	+
600 s	89703.44	91726.41	1002.96	103752.12	105322.90	682.78	100701.28	103535.95	1939.74	+
1800 s	88345.94	90382.56	935.31	103781.86	104967.31	509.08	96490.58	99862.35	1553.10	+

around 12 times more expensive than the construction of a solution with Options 1 or 2. Based on these results we decided for Option 2 for all further experiments.

A second set of experiments concerned the setting of the parameter r_{det}—the so-called *determinism rate*—which determines the percentage of deterministic versus probabilistic solution construction steps. We applied 10 versions of the ACO algorithm ($r_{\text{det}} \in \{0{,}0, 0{,}1, \ldots, 0{,}9\}$) with the fixed settings $n_a = 5$, Option 2, and the use of local search ($d = 3$) each 10 times for 600 seconds to the given problem instance. The results indicate a clear advantage of the algorithm versions that use medium determinism rates. With the intuition that at the beginning of a restart phase we need a higher determinism rate in order to reach good solutions quickly, each restart phase is started with $r_{\text{det}} = 0{,}7$. The determinism rate is then gradually decreased until $r_{\text{det}} = 0{,}3$ in each restart phase.

In a last set of tuning experiments we tested different settings for the number of allowed local search steps, that is, different settings for parameter d. We tested the 4 different settings $d \in \{1, 2, 3, 4\}$. The outcome (which is not shown for space reasons) was such that we used $d = 3$ for the final set of experiments.

2.3. Experimental results

In this section we present the results of ACO versus the results of the simple EA that was outlined in Section **??**. Additionally, we tested a version of our ACO approach in which the pheromone update procedure is not used (line 11 of Algorithm 1). This is often done in order to prove the usefulness of the learning component of ACO. The resulting ACO version is henceforth denoted by ACO*. We applied all three algorithms 30 times to our problem instance. This was done for 3 different time limits (120, 600, and 1800 seconds) in order to detect possible differences in the behaviour of the algorithms when different execution times are considered. The results are shown in numerical form in Table 3 and in graphical form in Figure 2. Since we are dealing with stochastic algorithms and we want to provide the results with statistical confidence, the following analysis has been performed. Firstly, a Kolmogorov-Smirnov test is performed in order to check whether the values of the results follow a normal (gaussian) distribution or not. If so, an ANOVA test is done, otherwise we perform a Kruskal-Wallis test. We always consider in this work a confidence level of 95 % (i.e., significance level of 5 % or p-value under 0,05) in the statistical tests, which means that the differences are unlikely to have occurred by chance with a probability of 95 %. Successful tests are marked with "+" symbols in the last column of Table 3; conversely, "−" means that no statistical confidence was found (p-value $> 0{,}05$).

We can observe that the ACO approach clearly outperforms the simple EA for what concerns all three time limits and with statistical confidence (see "+" symbols in the last column). Moreover, the ACO approach clearly outperforms the ACO* version, which means that the learning component of ACO is useful in this case. One main conclusion can be drawn from these results: the collaborative search plays a very important role here. Indeed, in ACO* no pheromone update is performed and therefore the constructive phase does not profit from the search experience, that is, previous ant paths, whereas in the (1,10) EA no recombination is required, so no genetic material is exchanged among individuals. In fact, it has been shown in the literature that well known crossover operators for EAs such as single point crossover do not perform well on this problem [4] and this is why $\mu = 1$ was chosen in [7] for the (μ, λ) EA.

Referencias

[1] C. Blum and M. Dorigo. The hyper-cube framework for ant colony optimization. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 34(2):1161–1172, 2004.

[2] R. Borndörfer, A. Eisenblätter, M. Grötschel, and A. Martin. Frequency assignment in cellular phone networks. *Annals of Operations Research*, 76:73 – 93, 1998.

[3] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT press, Cambridge, MA, 2004.

[4] R. Dorne and J.-K. Hao. An evolutionary approach for frequency assignment in cellular radio networks. In *Proc. of the IEEE Int. Conf. on Evolutionary Computation*, pages 539 – 544, 1995.

[5] FAP Web. http://fap.zib.de/.

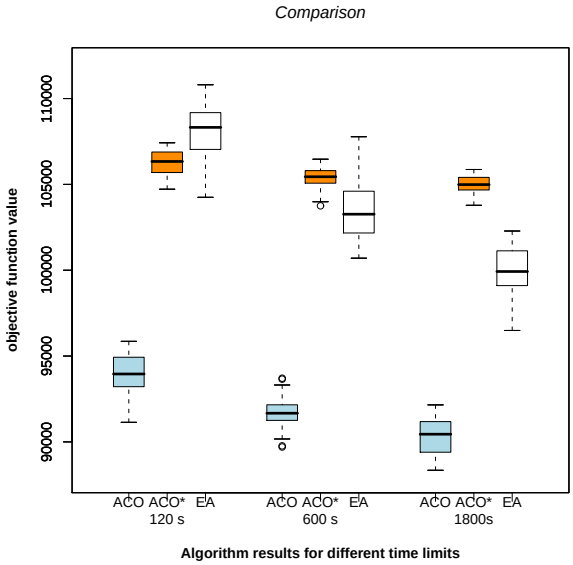

Figura 2: This graphic presents the results shown in Table 3 in the form of box-plots.

[6] A. M. J. Kuurne. On GSM mobile measurement based interference matrix generation. In *IEEE 55th Vehicular Technology Conference, VTC Spring 2002*, pages 1965 – 1969, 2002.

[7] F. Luna, E. Alba, A. J. Nebro, and S. Pedraza. Evolutionary algorithms for real-world instances of the automatic frequency planning problem in GSM networks. In *Seventh European Conference on Evolutionary Computation in Combinatorial Optimization (EVOCOP 2007)*, volume 4446 of *LNCS*, pages 108 – 120, 2007.

[8] V. Maniezzo and A. Carbonaro. An ANTS heuristic for the frequency assignment problem. *Future Generation Computer Systems*, 16(9):927–935, 2000.