Modelling parameterized shared-memory hyperheuristics for auto-tuning

José Matías Cutillas Lozano and Domingo Giménez
Departamento de Informática y Sistemas, University of Murcia
Luis Gabino Cutillas Lozano
Aguas Municipalizadas de Alicante

12th International Conference CMMSE - La Manga, July 2012

Contents

Hyperheuristics based on parameterized metaheuristics

- Selecting the appropriate values of parameters to apply a satisfactory metaheuristic to a particular problem can be difficult and is computationally demanding.
- Hyperheuristics based on a metaheuristic scheme (HMS) is used for selecting these values.
- Now the problem is to optimize the metaheuristic itself by the hyperheuristic.
- When executing the hyperheuristic many metaheuristics are applied. The execution time is very large and it is necessary to use parallelism.
- The application and auto-tuning of the hyperheuristic is analysed with a problem of cost optimization of electric consumption.

Hyperheuristics based on parameterized metaheuristics

- Selecting the appropriate values of parameters to apply a satisfactory metaheuristic to a particular problem can be difficult and is computationally demanding.
- Hyperheuristics based on a metaheuristic scheme (HMS) is used for selecting these values.
- Now the problem is to optimize the metaheuristic itself by the hyperheuristic.
- When executing the hyperheuristic many metaheuristics are applied. The execution time is very large and it is necessary to use parallelism.
- The application and auto-tuning of the hyperheuristic is analysed with a problem of cost optimization of electric consumption.

Hyperheuristics based on parameterized metaheuristics

- Selecting the appropriate values of parameters to apply a satisfactory metaheuristic to a particular problem can be difficult and is computationally demanding.
- Hyperheuristics based on a metaheuristic scheme (HMS) is used for selecting these values.
- Now the problem is to optimize the metaheuristic itself by the hyperheuristic.
- When executing the hyperheuristic many metaheuristics are applied. The execution time is very large and it is necessary to use parallelism.
- The application and auto-tuning of the hyperheuristic is analysed with a problem of cost optimization of electric consumption.

Hyperheuristics based on parameterized metaheuristics

- Selecting the appropriate values of parameters to apply a satisfactory metaheuristic to a particular problem can be difficult and is computationally demanding.
- Hyperheuristics based on a metaheuristic scheme (HMS) is used for selecting these values.
- Now the problem is to optimize the metaheuristic itself by the hyperheuristic.
- When executing the hyperheuristic many metaheuristics are applied. The execution time is very large and it is necessary to use parallelism.
- The application and auto-tuning of the hyperheuristic is analysed with a problem of cost optimization of electric consumption.

Hyperheuristics based on parameterized metaheuristics

- Selecting the appropriate values of parameters to apply a satisfactory metaheuristic to a particular problem can be difficult and is computationally demanding.
- Hyperheuristics based on a metaheuristic scheme (HMS) is used for selecting these values.
- Now the problem is to optimize the metaheuristic itself by the hyperheuristic.
- When executing the hyperheuristic many metaheuristics are applied. The execution time is very large and it is necessary to use parallelism.
- The application and auto-tuning of the hyperheuristic is analysed with a problem of cost optimization of electric consumption.

Parallel-parameterized metaheuristics and hyperheuristics

- The same parallelization techniques used in the metaheuristic scheme (MS) are applicable for hyperheuristics.
- A parallel metaheuristic is obtained by selecting the values of the metaheuristic parameters and the parallelism parameters for each function.
- Some auto-tuning technique is applied to select the optimum number of threads to obtain low execution times.
- Two basic parallel schemes are identified for the functions of the HMS and MS: one-level and two-level parallelism schemes.
- Although parallelism can be applied in the hyperheuristic and in the metaheuristics, it is usually applied only in the hyperheuristic (at a high level).

Parallel-parameterized metaheuristics and hyperheuristics

- The same parallelization techniques used in the metaheuristic scheme (MS) are applicable for hyperheuristics.
- A parallel metaheuristic is obtained by selecting the values of the metaheuristic parameters and the parallelism parameters for each function.
- Some auto-tuning technique is applied to select the optimum number of threads to obtain low execution times.
- Two basic parallel schemes are identified for the functions of the HMS and MS: one-level and two-level parallelism schemes.
- Although parallelism can be applied in the hyperheuristic and in the metaheuristics, it is usually applied only in the hyperheuristic (at a high level).

Parallel-parameterized metaheuristics and hyperheuristics

- The same parallelization techniques used in the metaheuristic scheme (MS) are applicable for hyperheuristics.
- A parallel metaheuristic is obtained by selecting the values of the metaheuristic parameters and the parallelism parameters for each function.
- Some auto-tuning technique is applied to select the optimum number of threads to obtain low execution times.
- Two basic parallel schemes are identified for the functions of the HMS and MS: one-level and two-level parallelism schemes.
- Although parallelism can be applied in the hyperheuristic and in the metaheuristics, it is usually applied only in the hyperheuristic (at a high level).

Parallel-parameterized metaheuristics and hyperheuristics

- The same parallelization techniques used in the metaheuristic scheme (MS) are applicable for hyperheuristics.
- A parallel metaheuristic is obtained by selecting the values of the metaheuristic parameters and the parallelism parameters for each function.
- Some auto-tuning technique is applied to select the optimum number of threads to obtain low execution times.
- Two basic parallel schemes are identified for the functions of the HMS and MS: one-level and two-level parallelism schemes.
- Although parallelism can be applied in the hyperheuristic and in the metaheuristics, it is usually applied only in the hyperheuristic (at a high level).

Parallel-parameterized metaheuristics and hyperheuristics

- The same parallelization techniques used in the metaheuristic scheme (MS) are applicable for hyperheuristics.
- A parallel metaheuristic is obtained by selecting the values of the metaheuristic parameters and the parallelism parameters for each function.
- Some auto-tuning technique is applied to select the optimum number of threads to obtain low execution times.
- Two basic parallel schemes are identified for the functions of the HMS and MS: one-level and two-level parallelism schemes.
- Although parallelism can be applied in the hyperheuristic and in the metaheuristics, it is usually applied only in the hyperheuristic (at a high level).

Modeling and auto-tuning

- We want to establish the optimum number of threads (which provides the minimum execution time) for each function of the metaheuristic.
- For this purpose, a model for each routine in the scheme is created.
- Constants of the model for the problem and system are obtained with the application of the auto-tuning technique.
- Auto-selecting the optimum number of threads as a function of the metaheuristic parameters is possible for each function.

Modeling and auto-tuning

- We want to establish the optimum number of threads (which provides the minimum execution time) for each function of the metaheuristic.
- For this purpose, a model for each routine in the scheme is created.
- Constants of the model for the problem and system are obtained with the application of the auto-tuning technique.
- Auto-selecting the optimum number of threads as a function of the metaheuristic parameters is possible for each function.

Modeling and auto-tuning

- We want to establish the optimum number of threads (which provides the minimum execution time) for each function of the metaheuristic.
- For this purpose, a model for each routine in the scheme is created.
- Constants of the model for the problem and system are obtained with the application of the auto-tuning technique.
- Auto-selecting the optimum number of threads as a function of the metaheuristic parameters is possible for each function.

Modeling and auto-tuning

- We want to establish the optimum number of threads (which provides the minimum execution time) for each function of the metaheuristic.
- For this purpose, a model for each routine in the scheme is created.
- Constants of the model for the problem and system are obtained with the application of the auto-tuning technique.
- Auto-selecting the optimum number of threads as a function of the metaheuristic parameters is possible for each function.

- Now the problem to optimize by the hyperheuristic is to obtain a satisfactory metaheuristic for the electricity consumption problem.
- In the hyperheuristic an individual or element is represented by an integer vector *MetaheurParam* of size 20 that encodes the set of parameters that characterizes a metaheuristic.
- The reference set of individuals is a set of metaheuristics with each metaheuristic or combination/hybridation of basic metaheuristics (GRASP, Scatter Search, Genetic algorithm and Tabu Search) given by the values in *MetaheurParam*.

- Now the problem to optimize by the hyperheuristic is to obtain a satisfactory metaheuristic for the electricity consumption problem.
- In the hyperheuristic an individual or element is represented by an integer vector *MetaheurParam* of size 20 that encodes the set of parameters that characterizes a metaheuristic.
- The reference set of individuals is a set of metaheuristics with each metaheuristic or combination/hybridation of basic metaheuristics (GRASP, Scatter Search, Genetic algorithm and Tabu Search) given by the values in *MetaheurParam*.

- Now the problem to optimize by the hyperheuristic is to obtain a satisfactory metaheuristic for the electricity consumption problem.
- In the hyperheuristic an individual or element is represented by an integer vector *MetaheurParam* of size 20 that encodes the set of parameters that characterizes a metaheuristic.
- The reference set of individuals is a set of metaheuristics with each metaheuristic or combination/hybridation of basic metaheuristics (GRASP, Scatter Search, Genetic algorithm and Tabu Search) given by the values in *MetaheurParam*.

- The fitness value in the hyperheuristic for an element *MetaheurParam* is the fitness value obtained when the metaheuristic with the parameters in *MetaheurParam* is applied to the electricity consumption problem.
- Our objective is to minimize the fitness function and so obtain the optimum combination of the metaheuristic parameters.
- Because an hyperheuristic executes a lot of metaheuristics, the execution time is very large and parallelism is necessary.
- To optimize the parallelism parameters, the same auto-tuning technique applied for metaheuristics is used.

- The fitness value in the hyperheuristic for an element *MetaheurParam* is the fitness value obtained when the metaheuristic with the parameters in *MetaheurParam* is applied to the electricity consumption problem.
- Our objective is to minimize the fitness function and so obtain the optimum combination of the metaheuristic parameters.
- Because an hyperheuristic executes a lot of metaheuristics, the execution time is very large and parallelism is necessary.
- To optimize the parallelism parameters, the same auto-tuning technique applied for metaheuristics is used.

- The fitness value in the hyperheuristic for an element *MetaheurParam* is the fitness value obtained when the metaheuristic with the parameters in *MetaheurParam* is applied to the electricity consumption problem.
- Our objective is to minimize the fitness function and so obtain the optimum combination of the metaheuristic parameters.
- Because an hyperheuristic executes a lot of metaheuristics, the execution time is very large and parallelism is necessary.
- To optimize the parallelism parameters, the same auto-tuning technique applied for metaheuristics is used.

- The fitness value in the hyperheuristic for an element *MetaheurParam* is the fitness value obtained when the metaheuristic with the parameters in *MetaheurParam* is applied to the electricity consumption problem.
- Our objective is to minimize the fitness function and so obtain the optimum combination of the metaheuristic parameters.
- Because an hyperheuristic executes a lot of metaheuristics, the execution time is very large and parallelism is necessary.
- To optimize the parallelism parameters, the same auto-tuning technique applied for metaheuristics is used.

Parallel-parametrized scheme

Initialize(S,ParamIni,ThreadsIni)
while (**not** EndCondition(S,ParamEnd,ThreadsEnd))
  SS = Select(S,ParamSel,ThreadsSel)
  $SS1$ = Combine(SS,ParamCom,ThreadsCom)
  $SS2$ = Improve($SS1$,ParamImp,ThreadsImp)
  S = Include($SS2$,ParamInc,ThreadsInc)

Independent parallelization of the functions,
with parallelism parameters (number of threads) for each function.
The optimum value of the parallelism parameters depends on the values of the metaheuristic parameters (the metaheuristic or combination of metaheuristics).

Metaheuristics

- Four pure metaheuristics: GRASP, Genetic algorithm (GA), Scatter search (SS), Tabu Search (TS)
- Eleven combinations: GRASP+GA, GRASP+SS, GRASP+TS, GA+SS, GA+TS, SS+TS, GRASP+GA+SS, GRASP+GA+TS, GRASP+SS+TS, GA+SS+TS, GRASP+GA+SS+TS
- Total: 15 metaheuristics or combinations of metaheuristics

Identify functions with the same parallel scheme:

One-level parallel scheme (scheme 1)

omp_set_num_threads($threads - one - level$(MetaheurParam))
#pragma omp parallel for
loop in elements
    treat element

i.e.: Initialize, Combine...

Two-level parallel scheme (scheme 2)

two–level(MetaheurParam) :
    omp_set_num_threads($threads - first - level$(MetaheurParam))
    #pragma omp parallel for
    loop in elements
        second–level(MetaheurParam,$threads - first - level$)

second–level(MetaheurParam,$threads - first - level$):
    omp_set_num_threads($threads - second - level$(MetaheurParam,$threads - first - level$))
    #pragma omp parallel for
    loop in neighbors
        treat neighbor

i.e.: Initialize, Improve...

Allows fine and coarse grained parallelism by changing the number of threads in each level

Modeling and auto-tuning

- An auto-tuning methodology is systematically applied to reduce execution time in the parallel-parameterized scheme.
- It has been applied to HMS and MS independently.
- It is necessary to select the values of the parallelism parameters (*ThreadsIni*, *ThreadsCom*, *ThreadsImp* and *ThreadsInc*) in the first and the second parallelism levels. A model of the execution time must be obtained for each function.
- Only HMS parallelism parameters are presented, but results are compared with those obtained for the MS.

Modeling and auto-tuning

- An auto-tuning methodology is systematically applied to reduce execution time in the parallel-parameterized scheme.
- It has been applied to HMS and MS independently.
- It is necessary to select the values of the parallelism parameters (*ThreadsIni*, *ThreadsCom*, *ThreadsImp* and *ThreadsInc*) in the first and the second parallelism levels. A model of the execution time must be obtained for each function.
- Only HMS parallelism parameters are presented, but results are compared with those obtained for the MS.

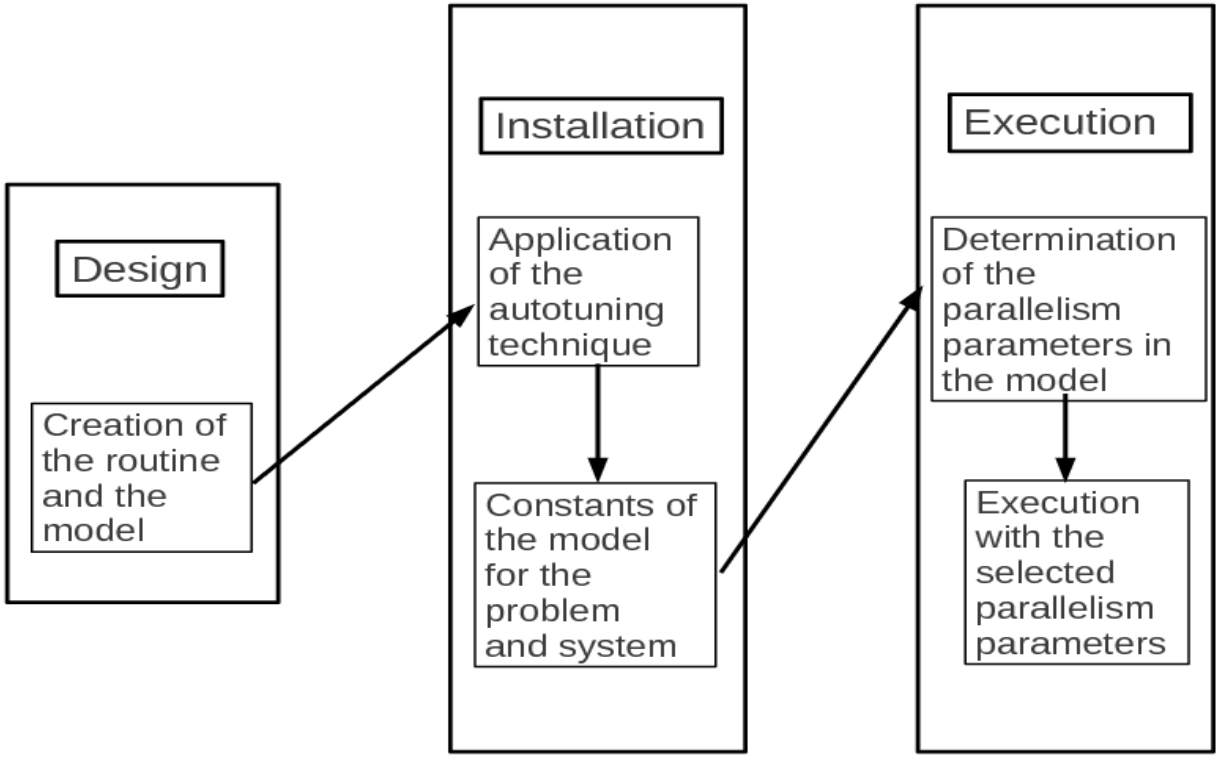
Modeling and auto-tuning

- An auto-tuning methodology is systematically applied to reduce execution time in the parallel-parameterized scheme.
- It has been applied to HMS and MS independently.
- It is necessary to select the values of the parallelism parameters (*ThreadsIni*, *ThreadsCom*, *ThreadsImp* and *ThreadsInc*) in the first and the second parallelism levels. A model of the execution time must be obtained for each function.
- Only HMS parallelism parameters are presented, but results are compared with those obtained for the MS.

Modeling and auto-tuning

- An auto-tuning methodology is systematically applied to reduce execution time in the parallel-parameterized scheme.
- It has been applied to HMS and MS independently.
- It is necessary to select the values of the parallelism parameters (*ThreadsIni*, *ThreadsCom*, *ThreadsImp* and *ThreadsInc*) in the first and the second parallelism levels. A model of the execution time must be obtained for each function.
- Only HMS parallelism parameters are presented, but results are compared with those obtained for the MS.

Scheme of the auto-tuning process



First phase: Design

The auto-tuning process has three phases:

- *First phase: Design.*
  - The routine is developed together with its theoretical execution time.
  - A model is obtained for each basic routine. Two basic models can be used, one for one-level routines (scheme 1) and another for nested parallelism (scheme 2).
  - The generation of the initial population in function Initialize with an initial number of elements in the reference set *INEIni* can be modeled:

$$t_{1-level} = \frac{k_g \cdot INEIni}{p} + k_p \cdot p \qquad (1)$$

Obtaining the model

- And the improvement of a percentage of the initial elements *PEIIni* with an intensification (extension of the considered neighborhood) *IIEIni* is modeled:

$$t_{2-levels} = \frac{k_i \cdot \frac{INEIni \cdot PEIIni \cdot IIEIni}{100}}{p_1 \cdot p_2} + k_{p,1} \cdot p_1 + k_{p,2} \cdot p_2 \qquad (2)$$

- For each of the other basic functions, the corresponding metaheuristic/hyperheuristic parameters are determined, and the model of the execution time is obtained as a function of those parameters and the parallelism parameters.

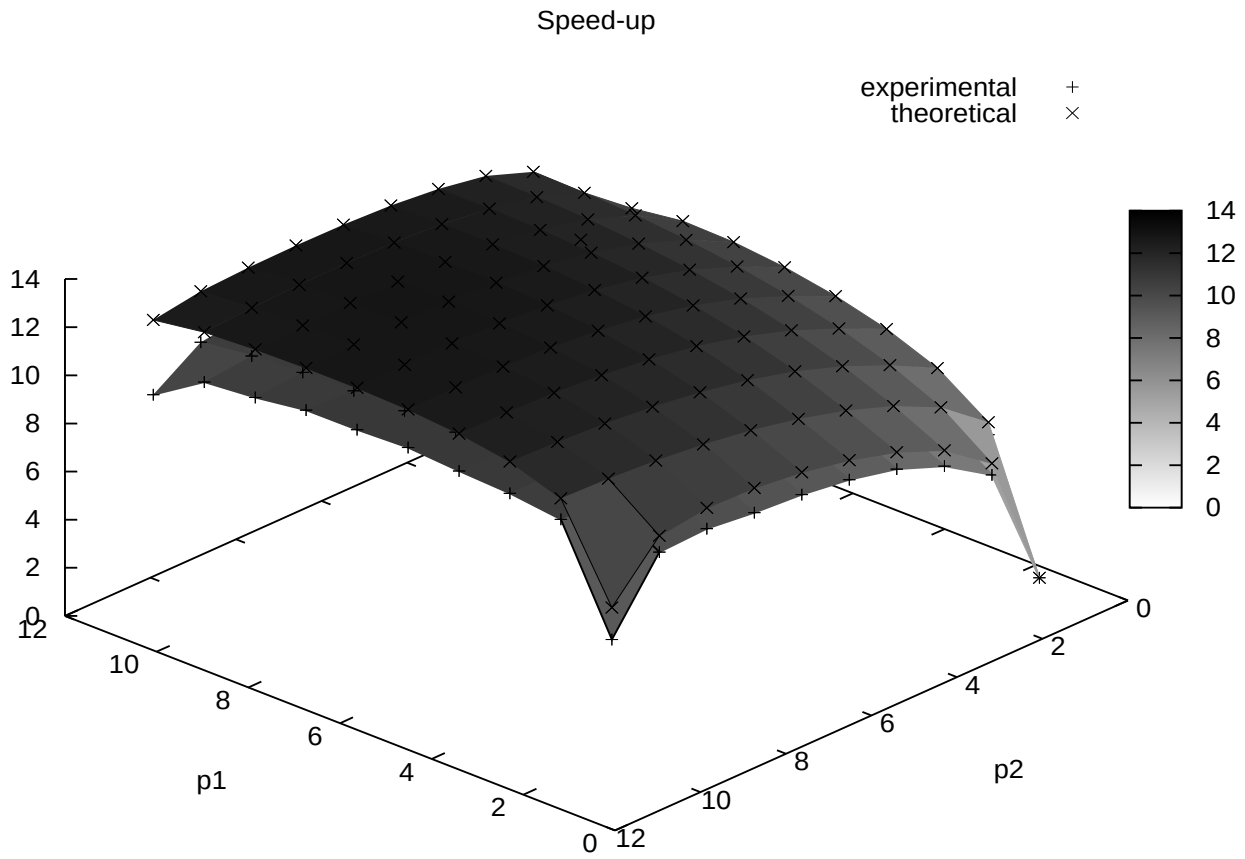
Second phase: Installation

- *Second phase: Installation.*
  - The parameters k_g, k_i, k_p, $k_{p,1}$ and $k_{p,2}$ used in the explanation of step 1 are estimated, as are the corresponding parameters for the other basic routines in the parallel scheme.
  - The optimum number of threads varies with the number of individuals, and we are interested in the selection at running time of a number of threads close to the optimum.
  - We summarize the results of the Installation of the scheme in an *HP Integrity Superdome SX2000* with 128 cores of *Intel Itanium-2 dual-core Montvale* with shared-memory.

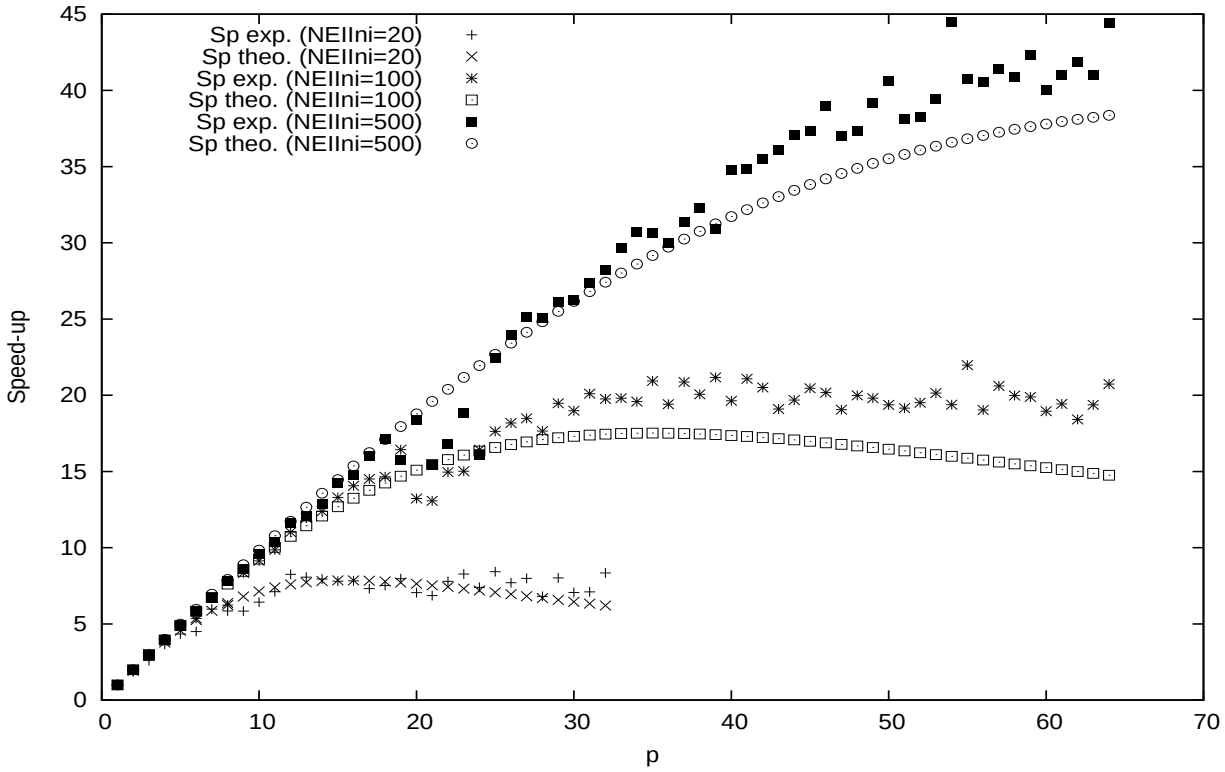
Experimental values of the model parameters

- For the one-level routine studied in the HMS, the values obtained by least-squares in the experiments with $INEIni = 5$ for the model in equation 1 are $k_g = 5.77 \cdot 10^{-1}$ and $k_p = 4.91 \cdot 10^{-2}$ (all in seconds).
- And for the two-level routine, the values of the other parallelism parameters for the model in equation 2 are $k_i = 1.21$, $k_{p,1} = 1.04 \cdot 10^{-1}$ and $k_{p,2} = 9.89 \cdot 10^{-2}$ seconds, obtained with hyperheuristic parameters $INEIni = 10$, $PEIIni = 100$, $IIEIni = 1$.
- The behavior of the routines in the system is well predicted.

Theoretical and experimental speed-ups when varying the number of threads of the first and second levels of parallelism for one parameter combination in a two-level parallel routine when applying the HMS.



Theoretical and experimental speed-up when varying the number of threads for three parameters in a one-level parallel routine when applying the MS (the behavior is well predicted).



Third phase: Execution

- *Third phase: Execution.*
  - At execution time the number of threads in each basic function is selected from the theoretical execution time (eqs. 1 and 2)
  - The number of threads which gives the theoretical mimimum execution time is obtained by minimizing the corresponding equation after substituting the values of the hyperheuristic and parallelism parameters.
  - For the initial generation of the reference set:

$$p_{opt.} = \sqrt{\frac{k_g}{k_p} \cdot INEIni} = 3.43 \cdot \sqrt{INEIni} \tag{3}$$

and for the improvement of the generated elements:

$$p_{1,opt.} = 4.79 \cdot 10^{-1} \cdot \sqrt[3]{INEIni \cdot PEIIni \cdot IIEIni} \tag{4}$$

$$p_{2,opt.} = 5.05 \cdot 10^{-1} \cdot \sqrt[3]{INEIni \cdot PEIIni \cdot IIEIni} \tag{5}$$

Execution results

To validate the auto-tuning methodology the optimum execution time and the speed-up for other values of the hyperheuristic parameters are compared using the parallelism parameters obtained in the Installation and the number of threads selected in the execution phase.

Table: Speed-up and number of threads for $INEIni = 20$ and 100 in the one-level parallel routine when applying the HMS. Optimum experimental values (optimum), values obtained with auto-tuning (model) and experimental speed-up values obtained from threads given by model (opt.-mod.).

	threads		speed-up		
INEIni	optimum	model	optimum	model	opt.-mod.
20	22	15	11	8	8
100	24	34	12	17	12

Execution results

Table: Speed-up and number of threads for the other two combinations of *INEIni*, *PEIIni* and *IIEIni* ((1): 50,50,1; (2): 100,50,1) in the two-level parallel routine when applying the HMS. Optimum experimental values (opt.), values obtained with auto-tuning (mod.) and experimental speed-up values obtained from threads given by model (opt.-mod.).

	threads 1-level		threads 2-levels		speed-up		
Comb.	opt.	mod.	opt.	mod.	opt.	mod.	opt.-mod.
(1)	9	6	8	7	14	15	11
(2)	9	8	4	9	15	24	14

The number of threads selected with the auto-tuning methodology is not far from the experimental optimum and, as a consequence, the speed-up achieved with auto-tuning is not far from the maximum.

Execution results

We can compare the results obtained for the hyperheuristic using the auto-tuning metodology with those achieved when directly applying individual metaheuristics to a problem of optimization of electrical costs.

Table: Speed-up and number of threads for $INEIni = 100$ and 500 in the one-level parallel routine when applying the MS. Optimum experimental values (opt.), values obtained with auto-tuning (mod.) and experimental speed-up values obtained from threads given by model (opt.-mod.).

	threads		speed-up		
INEIni	opt.	mod.	opt.	mod.	opt.-mod.
100	55	35	22	18	21
500	64	78	44	39	44

Execution results

Since the metaheuristic scheme is the same, similar results would be expected in both cases, although there may be differences due to different implementations.

Table: Speed-up and number of threads for other parameter combinations in the two-level parallel routine when applying the MS. Optimum experimental values (opt.), values obtained with auto-tuning (mod.) and experimental speed-up values obtained from threads given by model (opt.-mod.).

			threads		speed-up		
INEIni	*PEIIni*	*IIEIni*	opt.	mod.	opt.	mod.	opt.-mod.
100	50	10	30	26	15	11	13
500	100	5	32	59	29	27	29

Conclusions

- The auto-tuning methodology has been applied succesfully to parameterized shared-memory metaheuristic schemes and to hyperheuristics based on metaheuristic schemes.
- The applicability of the methodology has been shown with a problem of minimization of electricity consumption in wells exploitation and in a large shared-memory system.
- The methodology provides satisfactory values for the number of threads to use in the application of the parallel hyperheuristic, which has been shown for two basic functions in the hyperheuristic scheme, but the auto-tuning works in the same way for the other functions.

Future research

- Determine search ranges for each metaheuristic parameter, so reducing the possible values of the elements in the metaheuristic with which the hyperheuristic is implemented.
- Inclusion of more "pure" metaheuristics (ACO, PSO...).
- Develop unified parallel schemes for other computational systems (message-passing, hybrid, GPU...).