Modeling shared-memory metaheuristic schemes for autotuning

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

9th International Symposium DCAI - Salamanca, March 2012

Contents

Exploiting water resources

- The exploitation of water resources has a cost in the electricity consumption for pumping water.
- There are a number of technical constraints to be complied with.
- Our goal is to apply an algorithm that allows us to optimize the cost of electricity subject to the restrictions.
- An initial real problem solved using genetic algorithms will be improved.
- The space of possible solutions is very large and exhaustive methods are not applicable here.
- Metaheuristic techniques have been used.

Exploiting water resources

- The exploitation of water resources has a cost in the electricity consumption for pumping water.
- There are a number of technical constraints to be complied with.
- Our goal is to apply an algorithm that allows us to optimize the cost of electricity subject to the restrictions.
- An initial real problem solved using genetic algorithms will be improved.
- The space of possible solutions is very large and exhaustive methods are not applicable here.
- Metaheuristic techniques have been used.

Exploiting water resources

- The exploitation of water resources has a cost in the electricity consumption for pumping water.
- There are a number of technical constraints to be complied with.
- Our goal is to apply an algorithm that allows us to optimize the cost of electricity subject to the restrictions.
- An initial real problem solved using genetic algorithms will be improved.
- The space of possible solutions is very large and exhaustive methods are not applicable here.
- Metaheuristic techniques have been used.

Exploiting water resources

- The exploitation of water resources has a cost in the electricity consumption for pumping water.
- There are a number of technical constraints to be complied with.
- Our goal is to apply an algorithm that allows us to optimize the cost of electricity subject to the restrictions.
- An initial real problem solved using genetic algorithms will be improved.
- The space of possible solutions is very large and exhaustive methods are not applicable here.
- Metaheuristic techniques have been used.

Exploiting water resources

- The exploitation of water resources has a cost in the electricity consumption for pumping water.
- There are a number of technical constraints to be complied with.
- Our goal is to apply an algorithm that allows us to optimize the cost of electricity subject to the restrictions.
- An initial real problem solved using genetic algorithms will be improved.
- The space of possible solutions is very large and exhaustive methods are not applicable here.
- Metaheuristic techniques have been used.

Exploiting water resources

- The exploitation of water resources has a cost in the electricity consumption for pumping water.
- There are a number of technical constraints to be complied with.
- Our goal is to apply an algorithm that allows us to optimize the cost of electricity subject to the restrictions.
- An initial real problem solved using genetic algorithms will be improved.
- The space of possible solutions is very large and exhaustive methods are not applicable here.
- Metaheuristic techniques have been used.

Parallel-parameterized metaheuristics

- Experiments are carried out with various parameters and functions for tuning metaheuristics to our problem.
- Many experiments are required to select a good metaheuristic and to tune it to the problem.
- A large number of optimization problems will be solved.
- We use a unified parallel-parameterized scheme of metaheuristics:
  Different metaheuristics obtained from the parameterized scheme are parallelized with the parallel parameters for optimizing time execution.

Parallel-parameterized metaheuristics

- Experiments are carried out with various parameters and functions for tuning metaheuristics to our problem.
- Many experiments are required to select a good metaheuristic and to tune it to the problem.
- A large number of optimization problems will be solved.
- We use a unified parallel-parameterized scheme of metaheuristics:
  Different metaheuristics obtained from the parameterized scheme are parallelized with the parallel parameters for optimizing time execution.

Parallel-parameterized metaheuristics

- Experiments are carried out with various parameters and functions for tuning metaheuristics to our problem.
- Many experiments are required to select a good metaheuristic and to tune it to the problem.
- A large number of optimization problems will be solved.
- We use a unified parallel-parameterized scheme of metaheuristics:
  Different metaheuristics obtained from the parameterized scheme are parallelized with the parallel parameters for optimizing time execution.

Parallel-parameterized metaheuristics

- Experiments are carried out with various parameters and functions for tuning metaheuristics to our problem.
- Many experiments are required to select a good metaheuristic and to tune it to the problem.
- A large number of optimization problems will be solved.
- We use a unified parallel-parameterized scheme of metaheuristics:
  Different metaheuristics obtained from the parameterized scheme are parallelized with the parallel parameters for optimizing time execution.

Modeling and autotuning

- We want to establish the optimal number of threads (which provides the least execution time) for each function of the metaheuristic.
- Experiments are carried out varying the number of threads and parameters of each function, so correlating both variables.
- Auto-selecting the optimal number of threads for each function is possible with these correlations.

Modeling and autotuning

- We want to establish the optimal number of threads (which provides the least execution time) for each function of the metaheuristic.
- Experiments are carried out varying the number of threads and parameters of each function, so correlating both variables.
- Auto-selecting the optimal number of threads for each function is possible with these correlations.

Modeling and autotuning

- We want to establish the optimal number of threads (which provides the least execution time) for each function of the metaheuristic.
- Experiments are carried out varying the number of threads and parameters of each function, so correlating both variables.
- Auto-selecting the optimal number of threads for each function is possible with these correlations.

- The water system: a series of pumps (B) of known power, that draw water flow in their corresponding wells along a daily time range R.
- The pumps, that may be running or out of service at a given time, operate electrically and the electricity has a daily cost which should be minimized:

$$C_e = \sum_{i=1}^{R} \sum_{j=1}^{B} T_i P_j N_i X_{ij} \tag{1}$$

  - C_e: cost of the electricity consumed by the combination of pumps selected in a day.
  - T_i: cost of the electricity in the range i.
  - N_i: number of hours of pump operation in the time slot i.
  - P_j: electricity consumed by the pump j.
  - x_{ij}: has value 1 or 0 for pump on or off.

- Using the notation for evolutionary algorithms, an individual is represented by a binary vector of size $B \cdot R$ that encodes the set of pumps distributed in different time slots.
- Not all possible combinations result in feasible individuals, and each time an individual is generated or modified five constraints are evaluated:
  - Demand satisfaction.
  - Minimum flow maintenance.
  - Compliance with maximum exploitation volumes for each well.
  - Maintaining the average conductivity below the limit.
  - Compliance with maximum depths of dynamic levels.
- This means in some cases that obtaining a new individual is time-consuming. Furthermore, for large exploitation systems the number of wells and time ranges can be large.
- So, to apply different metaheuristics efficiently a shared-memory parameterized scheme is used, and the inclusion of an autotuning methodology in the scheme is studied.

- Using the notation for evolutionary algorithms, an individual is represented by a binary vector of size $B \cdot R$ that encodes the set of pumps distributed in different time slots.
- Not all possible combinations result in feasible individuals, and each time an individual is generated or modified five constraints are evaluated:
  - Demand satisfaction.
  - Minimum flow maintenance.
  - Compliance with maximum exploitation volumes for each well.
  - Maintaining the average conductivity below the limit.
  - Compliance with maximum depths of dynamic levels.
- This means in some cases that obtaining a new individual is time-consuming. Furthermore, for large exploitation systems the number of wells and time ranges can be large.
- So, to apply different metaheuristics efficiently a shared-memory parameterized scheme is used, and the inclusion of an autotuning methodology in the scheme is studied.

- Using the notation for evolutionary algorithms, an individual is represented by a binary vector of size $B \cdot R$ that encodes the set of pumps distributed in different time slots.
- Not all possible combinations result in feasible individuals, and each time an individual is generated or modified five constraints are evaluated:
  - Demand satisfaction.
  - Minimum flow maintenance.
  - Compliance with maximum exploitation volumes for each well.
  - Maintaining the average conductivity below the limit.
  - Compliance with maximum depths of dynamic levels.
- This means in some cases that obtaining a new individual is time-consuming. Furthermore, for large exploitation systems the number of wells and time ranges can be large.
- So, to apply different metaheuristics efficiently a shared-memory parameterized scheme is used, and the inclusion of an autotuning methodology in the scheme is studied.

- Using the notation for evolutionary algorithms, an individual is represented by a binary vector of size $B \cdot R$ that encodes the set of pumps distributed in different time slots.
- Not all possible combinations result in feasible individuals, and each time an individual is generated or modified five constraints are evaluated:
  - Demand satisfaction.
  - Minimum flow maintenance.
  - Compliance with maximum exploitation volumes for each well.
  - Maintaining the average conductivity below the limit.
  - Compliance with maximum depths of dynamic levels.
- This means in some cases that obtaining a new individual is time-consuming. Furthermore, for large exploitation systems the number of wells and time ranges can be large.
- So, to apply different metaheuristics efficiently a shared-memory parameterized scheme is used, and the inclusion of an autotuning methodology in the scheme is studied.

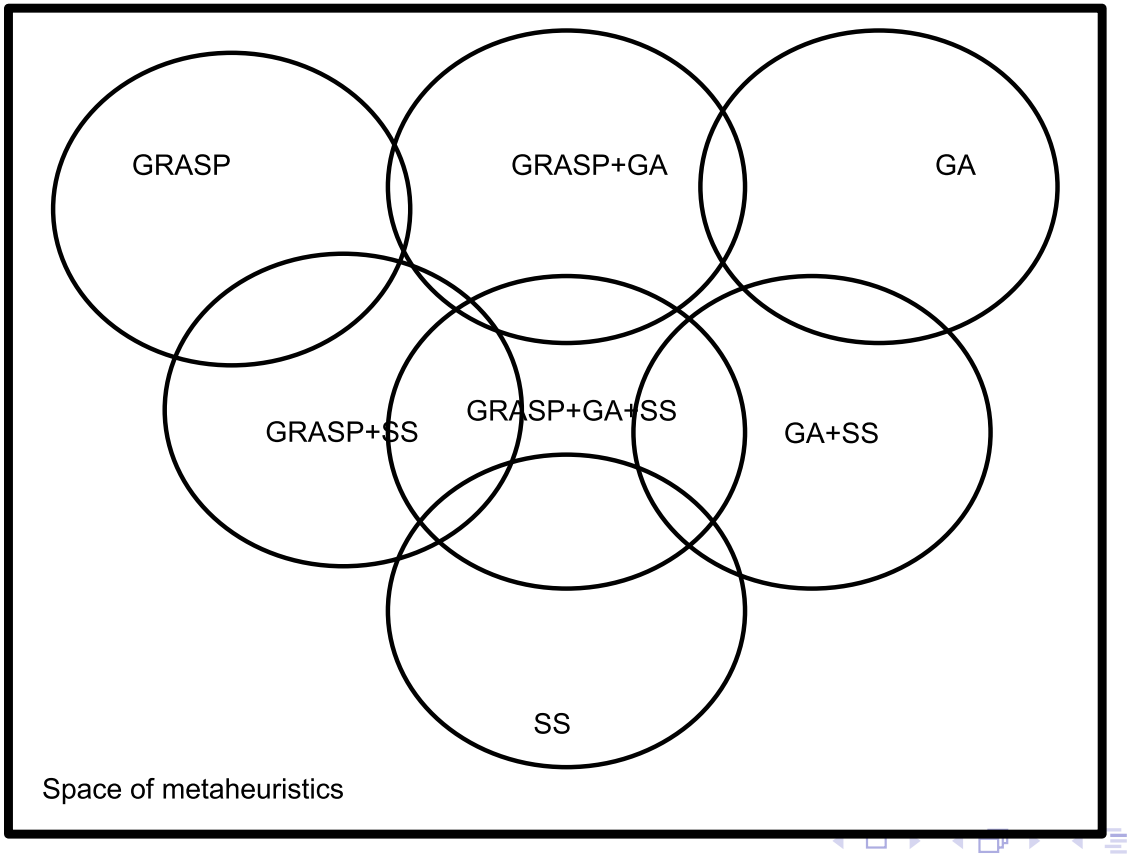
Parallel-parametrized scheme

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

- Pure metaheuristics: GRASP, Genetic algorithms (GA), Scatter search (SS)
- Combinations: GRASP+GA, GRASP+SS, GA+SS, GRASP+GA+SS



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

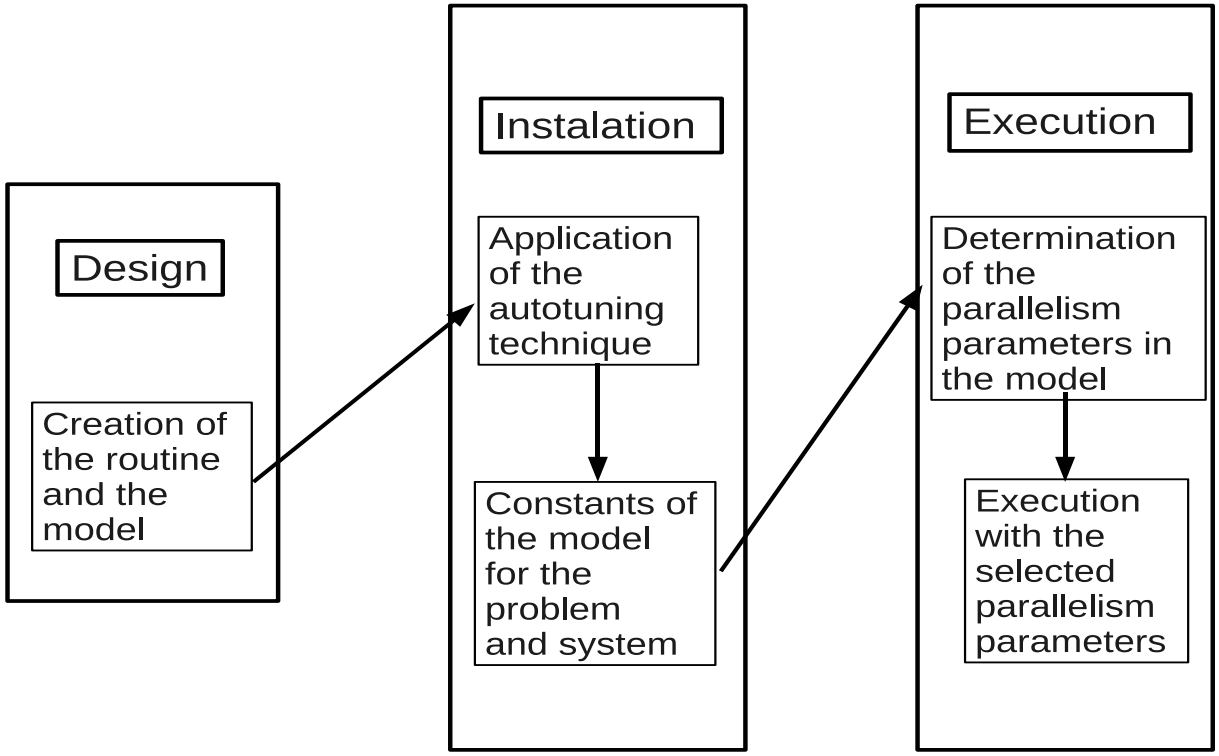
Modeling and autotuning

- An autotuning methodology is systematically applied to reduce execution time in the parallel-parameterized scheme.
- It is necessary to select appropriately 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.

Modeling and autotuning

- An autotuning methodology is systematically applied to reduce execution time in the parallel-parameterized scheme.
- It is necessary to select appropriately 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.

Scheme of the autotuning process



First phase: Design

The autotuning process has three phases:

- *First phase: Design.*
  - The routine is developed together with its theoretical execution time.
  - In the metaheuristic scheme, 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:

$$t_{1-level} = \frac{k_g \cdot INEIni}{p} + k_p \cdot p \qquad (2)$$

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:

$$t_{2-levels} = \frac{k_i \cdot \frac{INEIni \cdot PEIIni \cdot IIEIni}{100}}{p_1} + k_{p,1} \cdot p_1 + k_{p,2} \cdot p_2 \qquad (3)$$

- In our implementation, the second level is used to start more threads to work on the improvement of the fitness function (more neighbors are analyzed) but not to reduce the execution time.
- For each of the other basic functions, the corresponding metaheuristic parameters are determined, and the model of the execution time is obtained as a function of those parameters and the parallelism parameters.

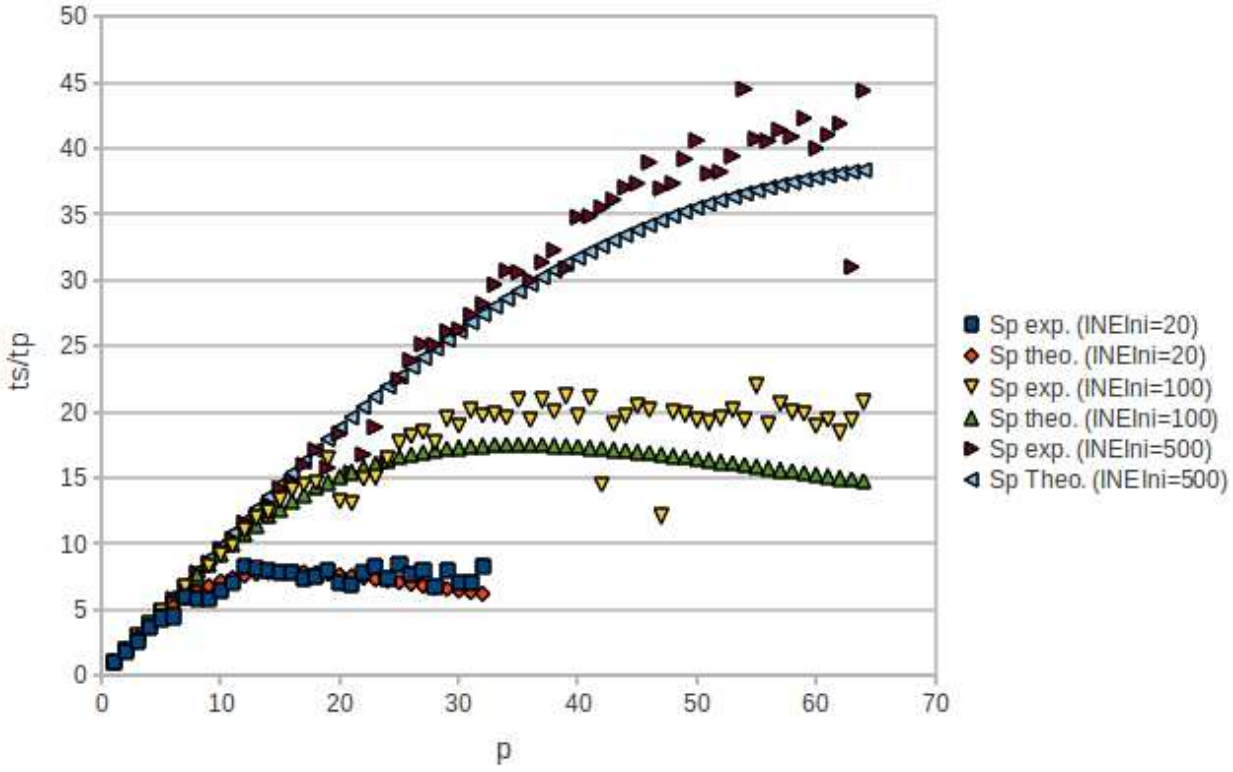
Second phase: Installation

- *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.
  - Because in our implementation the second level of parallelism is not used to reduce the execution time but to analyze a wider neighborhood, the parameter p_2 can also be considered as a parameter of the system-algorithm.
  - 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.

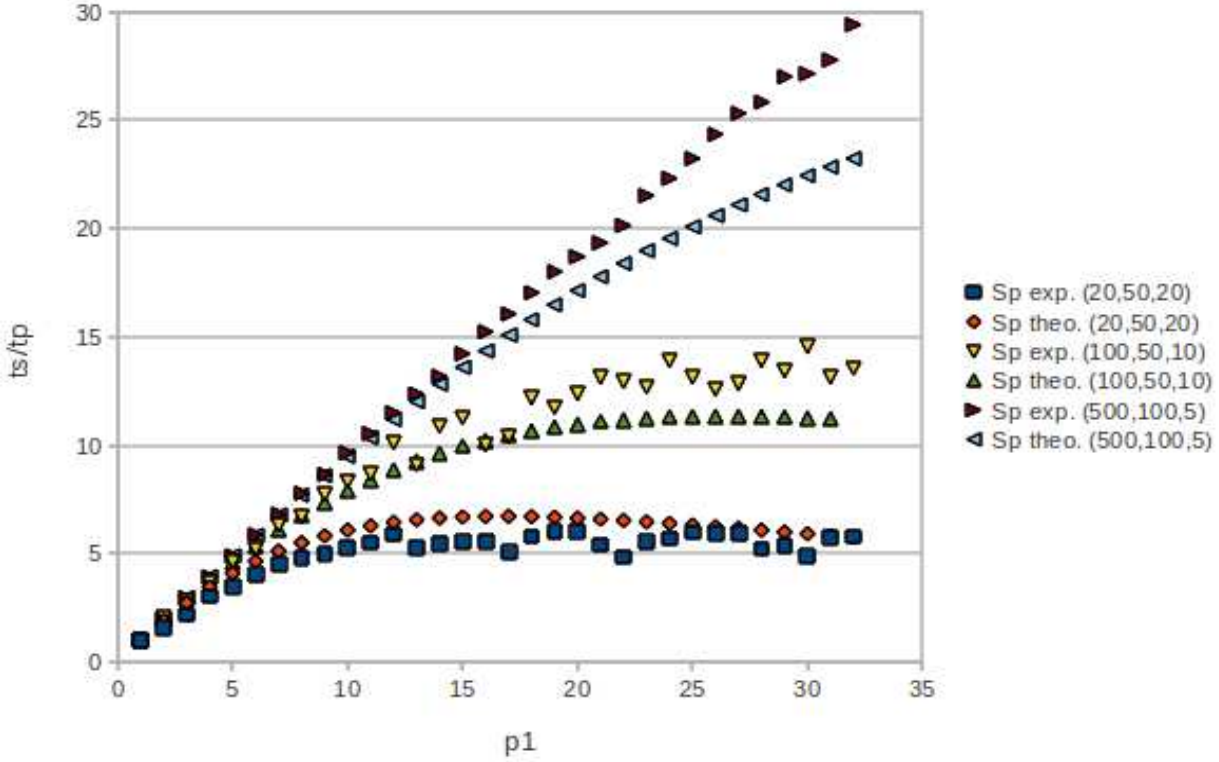
Experimental values of the model parameters

- 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.
- For the one-level routine studied, in the experiments with $INEIni = 20$ the values obtained by least-squares for the model in equation 2 are $k_g = 2.38 \cdot 10^{-3}$ and $k_p = 1.94 \cdot 10^{-4}$, all in seconds.
- For the two-level routine studied, the values of the other parallelism parameters for the model in equation 3 are $k_i = 9.10 \cdot 10^{-4}$, $k_{p,1} = 6.50 \cdot 10^{-4}$ and $k_{p,2} = 6.31 \cdot 10^{-3}$ seconds, obtained with metaheuristic parameters $INEIni = 20$, $PEIIni = 50$, $IIEIni = 20$ and a value of $p_2 = 1$.
- The behavior of the routines in the system are well predicted.

Theoretical and experimental speed-up for three parameters when varying the number of threads in the initial generation of the reference set



Theoretical and experimental speed-up for three combinations of the parameters *INEIni*, *PEIIni* and *IIEIni* when varying the number of threads in the improvement routine



Third phase: Execution

- *Third phase: Execution.*
  - At execution time the number of threads in each basic function is selected from the theoretical execution time (equations 2 and 3)
  - The number of threads which gives the theoretical mimimum execution time is obtained by minimizing the corresponding equation after substituting in it the values of the metaheuristic and parallelism parameters.
  - For the initial generation of the reference set:

$$p_{opt.} = \sqrt{\frac{k_g}{k_p} \cdot INEIni} = 3.50 \cdot \sqrt{INEIni} \tag{4}$$

and for the improvement of the generated elements:

$$p_{1,opt.} = 1.18 \cdot 10^{-1} \cdot \sqrt{INEIni \cdot PEIIni \cdot IIEIni} \tag{5}$$

Execution results

To validate the autotuning methodology the optimum execution time and the speed-up for other values of the metaheuristic 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 $INEIni = 100$ and 500 in the one-level parallel routine. Optimum experimental values (optimum) and values obtained with autotuning (model)

	threads		speed-up	
INEIni	optimum	model	optimum	model
100	55	35	22	18
500	64	78	44	39

Execution results

Table: Speed-up and number of threads for other parameter combinations in the two-level parallel routine. Optimum experimental values (optimum) and values obtained with autotuning (model)

			threads		speed-up	
INEIni	*PEIIni*	*IIEIni*	optimum	model	optimum	model
100	50	10	30	26	15	11
500	100	5	32	59	29	27

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

Conclusions

- An autotuning methodology has been adapted to obtain the number of threads to use in the application of a shared-memory parameterized metaheuristic scheme.
- It is shown that the autotuning methodology provides satisfactory values for the number of threads to use in the application of the parallel metaheuristic, working with a problem of minimization of electricity consumption and in a large shared-memory system.
- The methodology has been shown for two basic functions in the metaheuristic scheme, but the autotuning works the same way for the other functions.

Future research

- Inclusion of more "pure" metaheuristics (Tabu, Ant...).
- Design of hyperheuristics to automatically select the values of the metaheuristic parameters for a particular problem.
- Inclusion of autotuning in the parallel scheme, with some engine to autonomously select the number of threads.
- Develop unified parallel schemes for other computational systems (message-passing, hybrid, GPU...).