Application of metaheuristics to task-to-processors assignation problems

Domingo Giménez

Departamento de Informática y Sistemas
University of Murcia, Spain
domingo@um.es
http://dis.um.es/~domingo

Scheduling for large-scale systems, Knoxville, May 13-15 2009

Why metaheuristics?

- We use metaheuristics in different scientific problems
  … but we are not alone
- This presentation describes our experience with tasks mapping problems
- A common algorithmic scheme is used for the different metaheuristics
- A hierarchy of classes is proposed

Why metaheuristics?

- We use metaheuristics in different scientific problems
  … but we are not alone
- This presentation describes our experience with tasks mapping problems
- A common algorithmic scheme is used for the different metaheuristics
- A hierarchy of classes is proposed

Why metaheuristics?

- We use metaheuristics in different scientific problems
  … but we are not alone
- This presentation describes our experience with tasks mapping problems
- A common algorithmic scheme is used for the different metaheuristics
- A hierarchy of classes is proposed

Why metaheuristics?

- We use metaheuristics in different scientific problems
  ... but we are not alone
- This presentation describes our experience with tasks mapping problems
- A common algorithmic scheme is used for the different metaheuristics
- A hierarchy of classes is proposed

Contents

Presentation of the Parallel Computing Group - University of Murcia

Presentation of the Parallel Computing Group - University of Murcia

- Components
  - 2 doctors
  - 10 PhD students, from the:
    - Universidad Católica de Murcia
    - Centro de Supercomputación de Murcia
    - Marine studies company
    - Universidad Politécnica de Cartagena
    - Universidad Miguel Hernández de Elche
    - Universidade Federal do Estado da Bahia, Brazil
- Information
  - Group page: http://www.um.es/pcgum/
  - Publications: http://dis.um.es/~domingo/investigacion.html

Presentation of the Parallel Computing Group - University of Murcia

- Components
  - 2 doctors
  - 10 PhD students, from the:
    - Universidad Católica de Murcia
    - Centro de Supercomputación de Murcia
    - Marine studies company
    - Universidad Politécnica de Cartagena
    - Universidad Miguel Hernández de Elche
    - Universidade Federal do Estado da Bahia, Brazil
- Information
  - Group page:
    `http://www.um.es/pcgum/`
  - Publications:
    `http://dis.um.es/~domingo/investigacion.html`

Presentation of the Parallel Computing Group - University of Murcia

- Projects
  - Regional: **Adaptation and Optimization of Scientific Codes in Hierarchical Computational Systems**
    Collaboration with the Computational Electromagnetic group of the Universidad Politécnica de Cartagena
  - National: **Automatic Building and Optimization of Parallel Scientific Libraries**
    Collaboration with the universities: La Laguna, Jaume I of Castellón, Alicante, Politécnica de Valencia
  - Regional in preparation: **Solution of Biotechnology Problems with the Ben Arabí Supercomputer**
    Collaboration with the company Inbionova and the Plant pathology group

Presentation of the Parallel Computing Group - University of Murcia

- Applications
  - Orbits of artificial satellites - Observatorio Astronómico de la Armada Cádiz
  - Simulation of marine biosystems - Taxon Estudios Ambientales
  - Simultaneous equation models - Temporal series group, applications for medicine and psychology
  - Design of signal filters - Computational electromagnetic group
  - Physical engine of games - Centro de Supercomputación de Murcia
  - Biocatalizers - Inbionova
  - Cellular and molecular bases analysis - Plant pathology group
  - Regional meteorology simulations - Regional climate modelling group

Presentation of the Parallel Computing Group - University of Murcia

Metaheuristics

- Applications
  - Simultaneous equation models
    Automatic obtention of model from a set of data
  - Design of signal filters
    Design of the filter to obtain a given response function
  - Molecule simulation
    Estimation of the parameters to obtain the function which describes an experiment
- Tasks-to-processors assignation problems
  - To automatically optimize the execution of parallel routines
    - For parallel algorithmic schemes
    - For specific routines

Presentation of the Parallel Computing Group - University of Murcia

Metaheuristics

- Applications
  - Simultaneous equation models
    Automatic obtention of model from a set of data
  - Design of signal filters
    Design of the filter to obtain a given response function
  - Molecule simulation
    Estimation of the parameters to obtain the function which describes an experiment
- Tasks-to-processors assignation problems
  - To automatically optimize the execution of parallel routines
    - For parallel algorithmic schemes
    - For specific routines

Presentation of the Parallel Computing Group - University of Murcia

Metaheuristics

- Applications
  - Simultaneous equation models
    Automatic obtention of model from a set of data
  - Design of signal filters
    Design of the filter to obtain a given response function
  - Molecule simulation
    Estimation of the parameters to obtain the function which describes an experiment
- Tasks-to-processors assignation problems
  - To automatically optimize the execution of parallel routines
    - For parallel algorithmic schemes
    - For specific routines

Advantages of using metaheuristics

- General assignation problems are NP-complete
- Exact methods for specific problems, algorithms or systems
- In some cases the use of heuristics is satisfactory
- ... but in general it is not possible to obtain satisfactory assignations in a reduced time $\Longrightarrow$ metaheuristics
  - Provides a general framework for problems with different characteristics
    - Re-scheduling: new tasks, modifications in the system...
    - Hierarchical or distributed systems, on-chip systems...
  - Facilitates the development of different methods
  - Facilitates experimentation and tuning of the technique to the problem
  - Possible to combine different methods (hybridation)

Advantages of using metaheuristics

- General assignation problems are NP-complete
- Exact methods for specific problems, algorithms or systems
- In some cases the use of heuristics is satisfactory
- ... but in general it is not possible to obtain satisfactory assignations in a reduced time $\Longrightarrow$ metaheuristics
  - Provides a general framework for problems with different characteristics
    - Re-scheduling: new tasks, modifications in the system...
    - Hierarchical or distributed systems, on-chip systems...
  - Facilitates the development of different methods
  - Facilitates experimentation and tuning of the technique to the problem
  - Possible to combine different methods (hybridation)

Advantages of using metaheuristics

- General assignation problems are NP-complete
- Exact methods for specific problems, algorithms or systems
- In some cases the use of heuristics is satisfactory
- ... but in general it is not possible to obtain satisfactory assignations in a reduced time $\Longrightarrow$ metaheuristics
  - Provides a general framework for problems with different characteristics
    - Re-scheduling: new tasks, modifications in the system...
    - Hierarchical or distributed systems, on-chip systems...
  - Facilitates the development of different methods
  - Facilitates experimentation and tuning of the technique to the problem
  - Possible to combine different methods (hybridation)

General metaheuristic scheme

Use of a scheme common to different metaheuristics

while not EndCondition(S) **do**
  if $|SS| > 1$ **then**
    $SS1$=Combine(SS)
  else
    $SS1 = SS$
  end if
  $SS2$=Improve($SS1$)
  S=IncludeSolutions($SS2$)
end while

General metaheuristic scheme

- A common scheme
  - New versions of a metaheuristic just by modifying a function or a parameter in the scheme
  - A new metaheuristic just by modifying some functions or parameters in the scheme
  - Hybrid metaheuristics just by combining functions from different metaheuristics
- Possible to develop a hierarchy of classes to facilitate the development of metaheuristics
- But a different infrastructure is necessary for each assignation problem

General metaheuristic scheme

- A common scheme
  - New versions of a metaheuristic just by modifying a function or a parameter in the scheme
  - A new metaheuristic just by modifying some functions or parameters in the scheme
  - Hybrid metaheuristics just by combining functions from different metaheuristics
- Possible to develop a hierarchy of classes to facilitate the development of metaheuristics
- But a different infrastructure is necessary for each assignation problem

General metaheuristic scheme

- A common scheme
  - New versions of a metaheuristic just by modifying a function or a parameter in the scheme
  - A new metaheuristic just by modifying some functions or parameters in the scheme
  - Hybrid metaheuristics just by combining functions from different metaheuristics
- Possible to develop a hierarchy of classes to facilitate the development of metaheuristics
- But a different infrastructure is necessary for each assignation problem

General metaheuristic scheme

- A common scheme
  - New versions of a metaheuristic just by modifying a function or a parameter in the scheme
  - A new metaheuristic just by modifying some functions or parameters in the scheme
  - Hybrid metaheuristics just by combining functions from different metaheuristics
- Possible to develop a hierarchy of classes to facilitate the development of metaheuristics
- But a different infrastructure is necessary for each assignation problem

General metaheuristic scheme

- A common scheme
  - New versions of a metaheuristic just by modifying a function or a parameter in the scheme
  - A new metaheuristic just by modifying some functions or parameters in the scheme
  - Hybrid metaheuristics just by combining functions from different metaheuristics
- Possible to develop a hierarchy of classes to facilitate the development of metaheuristics
- But a different infrastructure is necessary for each assignation problem

Execution time model

- Our goal is to obtain the execution conditions which give the lowest execution time
- A model of the execution time is used. The model reflects:
  - The characteristics of the system
  - The possible modifications in the execution of the routine which should be selected to optimize the execution time

  if the model improves, the selection is better,
  but the methodology is the same independent of the model's accuracy

Execution time model

- Our goal is to obtain the execution conditions which give the lowest execution time
- A model of the execution time is used. The model reflects:
  - The characteristics of the system
  - The possible modifications in the execution of the routine

  which should be selected to optimize the execution time

  if the model improves, the selection is better,
  but the methodology is the same independent of the model's accuracy

Execution time model

- Our goal is to obtain the execution conditions which give the lowest execution time
- A model of the execution time is used. The model reflects:
  - The characteristics of the system
  - The possible modifications in the execution of the routine which should be selected to optimize the execution time

if the model improves, the selection is better,
but the methodology is the same independent of the model's accuracy

Execution time model

- Parameters to be obtained:
  - Block size of computations
  - Block size of communications
  - ...
  - number of processors
  - number of processes
  - logical topology of the processes
  - processes to processors mapping...
- A large number of parameters, for which a general determination method is not available
- and for different problems, different heuristics or metaheuristics are preferable

Execution time model

- Parameters to be obtained:
  - Block size of computations
  - Block size of communications
  - ...
  - number of processors
  - number of processes
  - logical topology of the processes
  - processes to processors mapping...
- A large number of parameters, for which a general determination method is not available
- and for different problems, different heuristics or metaheuristics are preferable

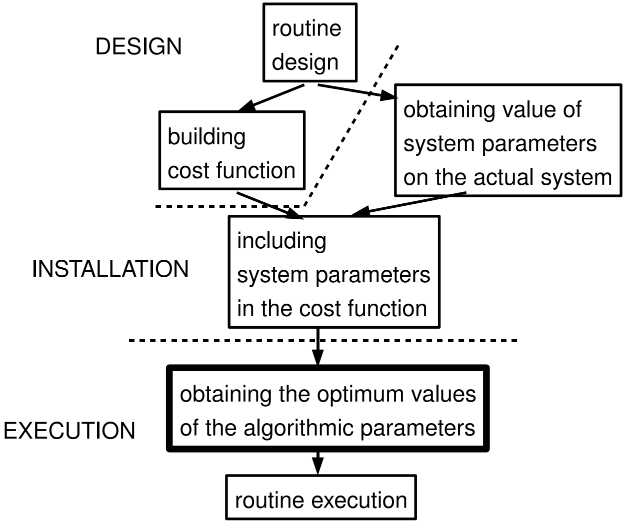
Execution time model

- Parameters to be obtained:
  - Block size of computations
  - Block size of communications
  - ...
  - number of processors
  - number of processes
  - logical topology of the processes
  - processes to processors mapping...
- A large number of parameters, for which a general determination method is not available
- and for different problems, different heuristics or metaheuristics are preferable

Execution time model

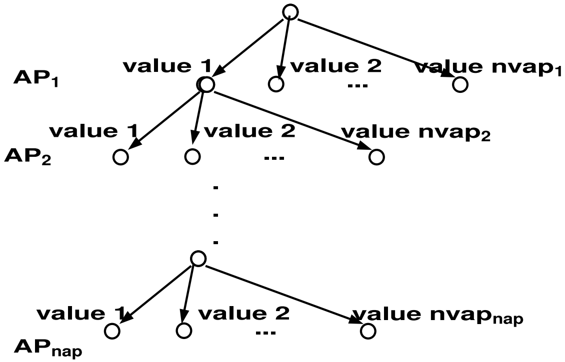
- Parameters to be obtained:
  - Block size of computations
  - Block size of communications
  - ...
  - number of processors
  - number of processes
  - logical topology of the processes
  - processes to processors mapping...
- A large number of parameters, for which a general determination method is not available
- and for different problems, different heuristics or metaheuristics are preferable

Optimization architecture



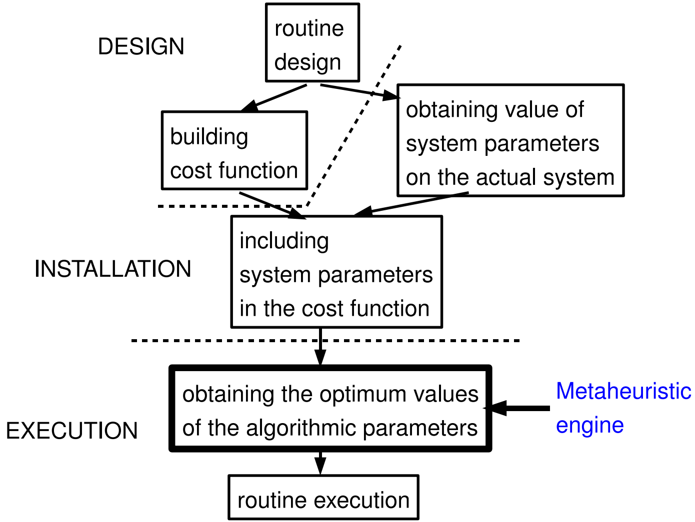
Decision tree

A large or huge decision tree



and it is impossible to use exact or analytical methods

Optimization architecture



Class hierarchy

- The development of a class hierarchy allows us to:
  - Reuse classes and methods
  - Add classes and methods
  - Develop new metaheuristics
  - Tune parameters and functions to the problem
  - Develop hybrid metaheuristics
  - Obtain a satisfactory metaheuristic for the problem
- but it is problem specific
- some typical problems could be included in the hierarchy
- and a tool to add new problems could be incorporated

Class hierarchy

- The development of a class hierarchy allows us to:
  - Reuse classes and methods
  - Add classes and methods
  - Develop new metaheuristics
  - Tune parameters and functions to the problem
  - Develop hybrid metaheuristics
  - Obtain a satisfactory metaheuristic for the problem
- but it is problem specific
- some typical problems could be included in the hierarchy
- and a tool to add new problems could be incorporated

Class hierarchy

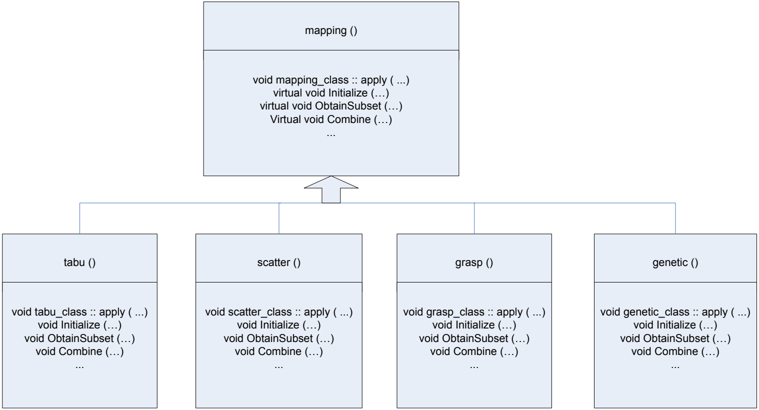
- The development of a class hierarchy allows us to:
  - Reuse classes and methods
  - Add classes and methods
  - Develop new metaheuristics
  - Tune parameters and functions to the problem
  - Develop hybrid metaheuristics
  - Obtain a satisfactory metaheuristic for the problem
- but it is problem specific
- some typical problems could be included in the hierarchy
- and a tool to add new problems could be incorporated

Class hierarchy

- The development of a class hierarchy allows us to:
  - Reuse classes and methods
  - Add classes and methods
  - Develop new metaheuristics
  - Tune parameters and functions to the problem
  - Develop hybrid metaheuristics
  - Obtain a satisfactory metaheuristic for the problem
- but it is problem specific
- some typical problems could be included in the hierarchy
- and a tool to add new problems could be incorporated

Classes hierarchy

As an example



General metaheuristic scheme

Use of a scheme common to different metaheuristics
possible reuse of functions

	Genetic	Scatter	GRASP	Tabu
Initialize	*	*	+	+
EndCondition	*	*	*	*
Combine	+	+		
Improve	-	+	+	-
Include	-	-	.	.

The problem

- An iterative scheme with computation and communication in each step
- A number of homogeneous processes: ideally the same volume of computation in each process
- Decide the number of processes and the number of processes assigned to each processor (d_i)
- The system is modelled with:
  - A vector t_c of costs of basic arithmetic operations on each processor
  - Two bidimensional arrays t_s and t_w, with the start and word-sending time between processes in two processors

The problem

- An iterative scheme with computation and communication in each step
- A number of homogeneous processes: ideally the same volume of computation in each process
- Decide the number of processes and the number of processes assigned to each processor (d_i)
- The system is modelled with:
  - A vector t_c of costs of basic arithmetic operations on each processor
  - Two bidimensional arrays t_s and t_w, with the start and word-sending time between processes in two processors

The problem

- An iterative scheme with computation and communication in each step
- A number of homogeneous processes: ideally the same volume of computation in each process
- Decide the number of processes and the number of processes assigned to each processor (d_i)
- The system is modelled with:
  - A vector t_c of costs of basic arithmetic operations on each processor
  - Two bidimensional arrays t_s and t_w, with the start and word-sending time between processes in two processors

The problem

- An iterative scheme with computation and communication in each step
- A number of homogeneous processes: ideally the same volume of computation in each process
- Decide the number of processes and the number of processes assigned to each processor (d_i)
- The system is modelled with:
    - A vector t_c of costs of basic arithmetic operations on each processor
    - Two bidimensional arrays t_s and t_w, with the start and word-sending time between processes in two processors

Execution time model

- No overlapping of computation and communication is considered: $t_{comp} + t_{comm}$
- The arithmetic cost is modelled: $t_{comp} = max_{i=1,\dots,P}\left\{n_{comp}(i)t_{c_i}\right\}$
- and the communication cost: $max_{i=1,\dots,P;\ j=1,\dots,P}\left\{n_{str}(i,j)t_{s_{ij}}\right\} + max_{i=1,\dots,P;\ j=1,\dots,P}\left\{n_{dat}(i,j)t_{w_{ij}}\right\}$
- … but other models could be considered

Execution time model

- No overlapping of computation and communication is considered: $t_{comp} + t_{comm}$
- The arithmetic cost is modelled:
  $t_{comp} = max_{i=1,\dots,P}\left\{n_{comp}(i) t_{c_i}\right\}$
- and the communication cost:
  $max_{i=1,\dots,P;\ j=1,\dots,P}\left\{n_{str}(i,j) t_{s_{ij}}\right\} +$
  $max_{i=1,\dots,P;\ j=1,\dots,P}\left\{n_{dat}(i,j) t_{w_{ij}}\right\}$
- ... but other models could be considered

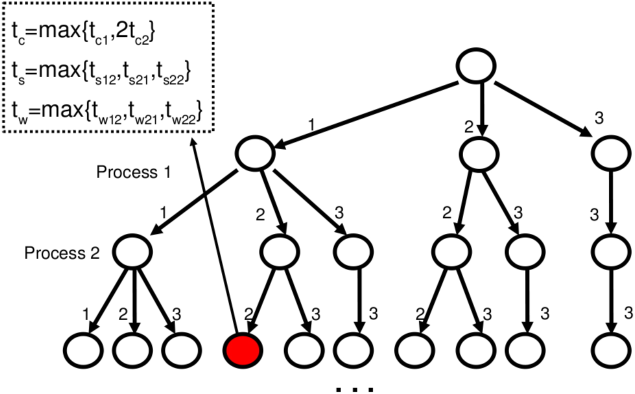
Execution time model

- No overlapping of computation and communication is considered: $t_{comp} + t_{comm}$
- The arithmetic cost is modelled: $t_{comp} = max_{i=1,\dots,P}\left\{n_{comp}(i)t_{c_i}\right\}$
- and the communication cost: $max_{i=1,\dots,P;\ j=1,\dots,P}\left\{n_{str}(i,j)t_{s_{ij}}\right\} + max_{i=1,\dots,P;\ j=1,\dots,P}\left\{n_{dat}(i,j)t_{w_{ij}}\right\}$
- … but other models could be considered

Execution time model

- No overlapping of computation and communication is considered: $t_{comp} + t_{comm}$
- The arithmetic cost is modelled:
  $t_{comp} = max_{i=1,\ldots,P}\left\{n_{comp}(i)t_{c_i}\right\}$
- and the communication cost:
  $max_{i=1,\ldots,P;\ j=1,\ldots,P}\left\{n_{str}(i,j)t_{s_{ij}}\right\} +$
  $max_{i=1,\ldots,P;\ j=1,\ldots,P}\left\{n_{dat}(i,j)t_{w_{ij}}\right\}$
- ... but other models could be considered

Assignation tree



Metaheuristics

- Metaheuristics experimented with:
  - Hill climbing
  - Tabu search
  - Scatter search
  - Genetic algorithms
  - Ant colony
  - Simulated annealing
- And exact methods with heuristics or probability:
  - Backtracking and Branch and Bound with pruning based on heuristics (possibly pruning nodes which could lead to the optimum solution) and tree traversal guided by heuristics
  - Probabilistic algorithms

Metaheuristics

- Metaheuristics experimented with:
  - Hill climbing
  - Tabu search
  - Scatter search
  - Genetic algorithms
  - Ant colony
  - Simulated annealing
- And exact methods with heuristics or probability:
  - Backtracking and Branch and Bound with pruning based on heuristics (possibly pruning nodes which could lead to the optimum solution) and tree traversal guided by heuristics
  - Probabilistic algorithms

Experience

- Easy development of metaheuristics
- Reuse of functions
- Easy development of parallel metaheuristics
- Application to dynamic programming and LU factorization
- Similar results for the different metaheuristics
- Better results (lower theoretical execution time and lower decision time) with metaheuristics

Experience

- Easy development of metaheuristics
- Reuse of functions
- Easy development of parallel metaheuristics
- Application to dynamic programming and LU factorization
- Similar results for the different metaheuristics
- Better results (lower theoretical execution time and lower decision time) with metaheuristics

Experience

- Easy development of metaheuristics
- Reuse of functions
- Easy development of parallel metaheuristics
- Application to dynamic programming and LU factorization
- Similar results for the different metaheuristics
- Better results (lower theoretical execution time and lower decision time) with metaheuristics

Experience

- Easy development of metaheuristics
- Reuse of functions
- Easy development of parallel metaheuristics
- Application to dynamic programming and LU factorization
- Similar results for the different metaheuristics
- Better results (lower theoretical execution time and lower decision time) with metaheuristics

Experience

- Easy development of metaheuristics
- Reuse of functions
- Easy development of parallel metaheuristics
- Application to dynamic programming and LU factorization
- Similar results for the different metaheuristics
- Better results (lower theoretical execution time and lower decision time) with metaheuristics

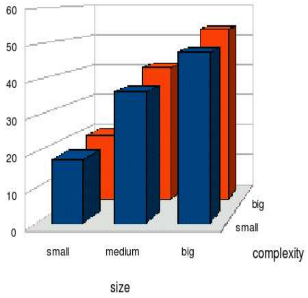
Experience

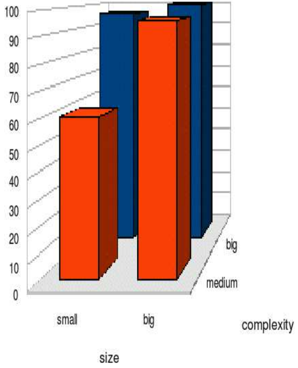
- Easy development of metaheuristics
- Reuse of functions
- Easy development of parallel metaheuristics
- Application to dynamic programming and LU factorization
- Similar results for the different metaheuristics
- Better results (lower theoretical execution time and lower decision time) with metaheuristics

Comparison of Backtacking and Scatter Search

Percentage of runs in which Scatter Search obtains better Total Time (Modelled Time plus Decision Time) than backtracking with node pruning

real system with 6 processors ; simulated system with 60 processors





The problem

- A master processor generates a set of independent tasks that are solved by slave processors
- Each task has certain memory requirements and each processor has a certain amount of memory
- The assignation of tasks to slave processors is done statically: $d_i = j$ means task i is assigned to processor j
- Each processor recives a new task when it has processed the one previously assigned
- The goal is to obtain the assignation with lowest theoretical time

The problem

- A master processor generates a set of independent tasks that are solved by slave processors
- Each task has certain memory requirements and each processor has a certain amount of memory
- The assignation of tasks to slave processors is done statically: $d_i = j$ means task i is assigned to processor j
- Each processor recives a new task when it has processed the one previously assigned
- The goal is to obtain the assignation with lowest theoretical time

The problem

- A master processor generates a set of independent tasks that are solved by slave processors
- Each task has certain memory requirements and each processor has a certain amount of memory
- The assignation of tasks to slave processors is done statically: $d_i = j$ means task i is assigned to processor j
- Each processor recives a new task when it has processed the one previously assigned
- The goal is to obtain the assignation with lowest theoretical time

The problem

- A master processor generates a set of independent tasks that are solved by slave processors
- Each task has certain memory requirements and each processor has a certain amount of memory
- The assignation of tasks to slave processors is done statically: $d_i = j$ means task i is assigned to processor j
- Each processor recives a new task when it has processed the one previously assigned
- The goal is to obtain the assignation with lowest theoretical time

The problem

- A master processor generates a set of independent tasks that are solved by slave processors
- Each task has certain memory requirements and each processor has a certain amount of memory
- The assignation of tasks to slave processors is done statically: $d_i = j$ means task i is assigned to processor j
- Each processor recives a new task when it has processed the one previously assigned
- The goal is to obtain the assignation with lowest theoretical time

Execution time model

- Number of basic operations for each task: c_i, $i = 1, \ldots, T$
- Given assignation d, the cost in processor j: $t_{c_j} \sum_{l=1,d_l=j}^{T} c_j$
- For an assignation d the cost is: $\max_{j=1,\ldots,P} \left\{ t_{c_j} \sum_{l=1,d_l=j}^{T} c_j \right\}$
- The optimization problem:
  $\min_d \left\{ \max_{j=1,\ldots,P} \left\{ t_{c_j} \sum_{l=1,d_l=j}^{T} c_j \right\} \right\}$

Execution time model

- Number of basic operations for each task: c_i, $i = 1, \ldots, T$
- Given assignation d, the cost in processor j: $t_{c_j} \sum_{l=1, d_l=j}^{T} c_j$
- For an assignation d the cost is: $\max_{j=1,\ldots,P} \left\{ t_{c_j} \sum_{l=1, d_l=j}^{T} c_j \right\}$
- The optimization problem: $\min_d \left\{ \max_{j=1,\ldots,P} \left\{ t_{c_j} \sum_{l=1, d_l=j}^{T} c_j \right\} \right\}$

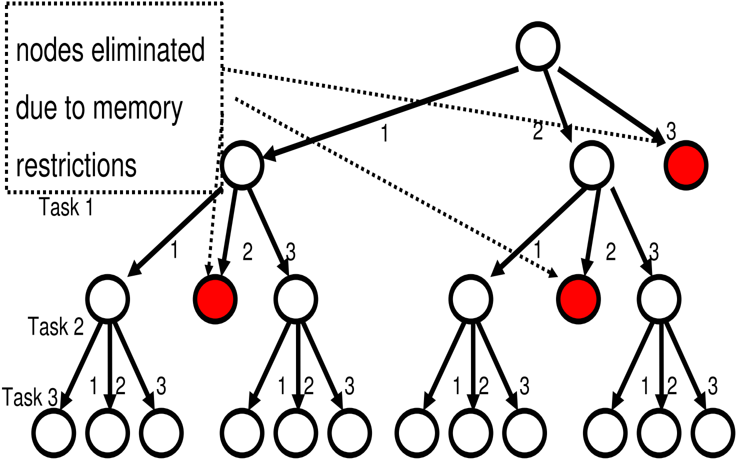
Execution time model

- Number of basic operations for each task: c_i, $i = 1, \ldots, T$
- Given assignation d, the cost in processor j: $t_{c_j} \sum_{l=1,d_l=j}^{T} c_j$
- For an assignation d the cost is: $\max_{j=1,\ldots,P} \left\{ t_{c_j} \sum_{l=1,d_l=j}^{T} c_j \right\}$
- The optimization problem: $\min_d \left\{ \max_{j=1,\ldots,P} \left\{ t_{c_j} \sum_{l=1,d_l=j}^{T} c_j \right\} \right\}$

Execution time model

- Number of basic operations for each task: c_i, $i = 1, \dots, T$
- Given assignation d, the cost in processor j: $t_{c_j} \sum_{l=1, d_l=j}^{T} c_j$
- For an assignation d the cost is: $\max_{j=1,\dots,P} \left\{ t_{c_j} \sum_{l=1, d_l=j}^{T} c_j \right\}$
- The optimization problem:
  $\min_d \left\{ \max_{j=1,\dots,P} \left\{ t_{c_j} \sum_{l=1, d_l=j}^{T} c_j \right\} \right\}$

Assignation tree



Use of the class hierarchy

- Metaheuristics:
  - Tabu search
  - Scatter search
  - Genetic algorithms
  - GRASP
- Integrated in the hierarchy
- Common problem and solution classes
- Reutilization of functions
- Easy tuning to the problem
- Hybridation

Use of the class hierarchy

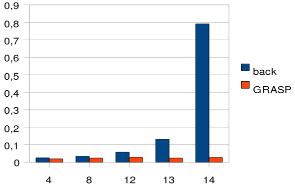
- Metaheuristics:
  - Tabu search
  - Scatter search
  - Genetic algorithms
  - GRASP
- Integrated in the hierarchy
- Common problem and solution classes
- Reutilization of functions
- Easy tuning to the problem
- Hybridation

Example of function reutilization

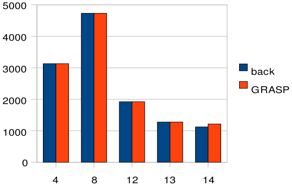
- The greedy functions used in GRASP can be used in
  - Scatter Search, in the improvement of the elements, which is done after each element is generated
  - Genetic Algorithms, in the individual generated in the mutation, so allowing the descendant of the individual to survive some generations and contribute to improve the population
  - Tabu Search, to improve the best element

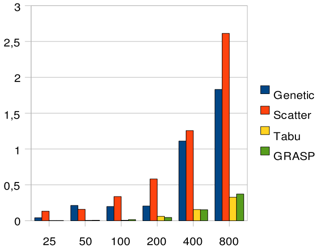
Comparison of metaheuristics

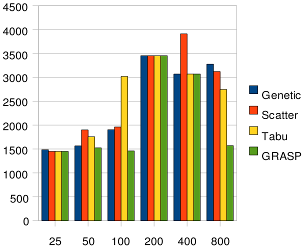
Mapping time



modelled time







The problem

- A master processor generates subproblems up to a certain level
- and assign the subproblems to the slave processors.
  The assignation can be:
  - Contiguous
  - Cyclic
  - Solving an optimization problem by using metaheuristics,
    ... but the main difficulty is in modelling the execution time because we do not know the number of nodes which will be generated
    and it depends on the problem but also on the input

The problem

- A master processor generates subproblems up to a certain level
- and assign the subproblems to the slave processors.
  The assignation can be:
  - Contiguous
  - Cyclic
  - Solving an optimization problem by using metaheuristics,
    ... but the main difficulty is in modelling the execution time because we do not know the number of nodes which will be generated
    and it depends on the problem but also on the input

Execution time model

- The execution time is knc, with:
  - n number of nodes in the tree
  - c the cost of evaluation of each node, estimated at installation time
  - k the percentage of nodes generated, estimated at installation time for some representative inputs and updated at running time with a subproblem of the problem to be solved
- Different possibilities to estimate and update k
- Other possible theoretical models, but with the same problem

Execution time model

- The execution time is knc, with:
  - n number of nodes in the tree
  - c the cost of evaluation of each node, estimated at installation time
  - k the percentage of nodes generated, estimated at installation time for some representative inputs and updated at running time with a subproblem of the problem to be solved
- Different possibilities to estimate and update k
- Other possible theoretical models, but with the same problem

Execution time model

- The execution time is knc, with:
  - n number of nodes in the tree
  - c the cost of evaluation of each node, estimated at installation time
  - k the percentage of nodes generated, estimated at installation time for some representative inputs and updated at running time with a subproblem of the problem to be solved
- Different possibilities to estimate and update k
- Other possible theoretical models, but with the same problem

Execution time model

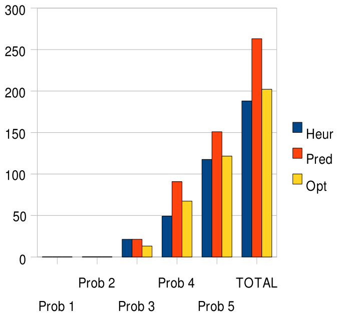
- The execution time is knc, with:
  - n number of nodes in the tree
  - c the cost of evaluation of each node, estimated at installation time
  - k the percentage of nodes generated, estimated at installation time for some representative inputs and updated at running time with a subproblem of the problem to be solved
- Different possibilities to estimate and update k
- Other possible theoretical models, but with the same problem

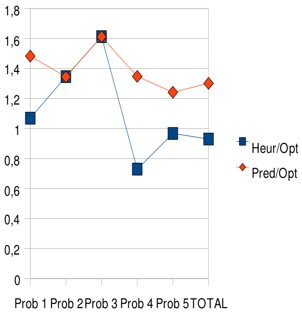
Execution time model

- The execution time is knc, with:
  - n number of nodes in the tree
  - c the cost of evaluation of each node, estimated at installation time
  - k the percentage of nodes generated, estimated at installation time for some representative inputs and updated at running time with a subproblem of the problem to be solved
- Different possibilities to estimate and update k
- Other possible theoretical models, but with the same problem

Preliminary results

Knapsack 0/1, problems of size 40. `Heur` obtention of the assignation with a greedy method, `Pred` with predetermined values from experiments with smaller problems, `Opt` the lowest experimental time obtained from executions varying the parameters but without approximating the optimization problem.





Conclusions

- Metaheuristics are useful to solve mapping problems
- We proposse to use a unified approach with a common algorithmic scheme and a class hierarchy
- This reduces programming and experiment costs
- But for each problem a different problem class and different functions
- It is necessary to develop a tool to facilitate the inclusion of new problems
- Until now preliminary results with some mapping problems

Conclusions

- Metaheuristics are useful to solve mapping problems
- We proposse to use a unified approach with a common algorithmic scheme and a class hierarchy
- This reduces programming and experiment costs
- But for each problem a different problem class and different functions
- It is necessary to develop a tool to facilitate the inclusion of new problems
- Until now preliminary results with some mapping problems

Conclusions

- Metaheuristics are useful to solve mapping problems
- We proposse to use a unified approach with a common algorithmic scheme and a class hierarchy
- This reduces programming and experiment costs
- But for each problem a different problem class and different functions
- It is necessary to develop a tool to facilitate the inclusion of new problems
- Until now preliminary results with some mapping problems

and credits

- Metaheuristic scheme: Francisco Almeida, Juan-Pedro Martínez-Gallar
- Class hierarchy: Javier Cuenca, Antonio Llanes
- Iterative scheme: Ángel-Luis Calvo, Ana Cortés, Juan-Pedro Martínez-Gallar, Carmela Pozuelo
- Master-slave: Javier Cuenca
- Backtracking: Manuel Quesada
- Translation: Stephen Hasler

... and more

Application of metaheuristics to task-to-processors assignation problems

Domingo Giménez

Departamento de Informática y Sistemas
University of Murcia, Spain
domingo@um.es
http://dis.um.es/~domingo

Scheduling for large-scale systems, Knoxville, May 13-15 2009