Using Parallelism in Experimenting and Fine Tuning of Parameters for Metaheuristics
Blesa, M.J. and Xhafa, F.
Research Report LSI-03-56-R
Using Parallelism in Experimenting and Fine Tuning of Parameters for Metaheuristics*

Maria Blesa and Fatos Khafa

Dept. Lleugatges i Sistemes Informàtics
Universitat Politècnica de Catalunya,
C6 Campus Nord, E-08034 Barcelona, Spain
{mjblesa,fatos}@lsi.upc.es

Abstract. We address the issue of using parallel implementations as a mean for efficient experimenting and fine tuning of parameters for metaheuristics—a very difficult and especially time consuming process. This is particularly of great impact to the metaheuristics, at least for two reasons: (a) measuring the performance of a metaheuristic implementation requires testing on a large set of instances and on real world instances usually of big and very big size: a considerable amount of time is needed to accomplish it; (b) the finding of right values for the search parameters of the metaheuristic is almost indispensable for the success of the metaheuristic implementation. Due to this, considerable efforts have been done by researchers and practitioners to provide, on the one hand, a methodology and rigorous basis for experimental evaluation of heuristics (e.g. Rardin and Uszay [22], Barr et al. [5]) and, on the other, to find efficient approaches for fine tuning of parameters such as developing specific software (e.g. Addesso-Dax and Laguna [1]), use of experimental design (e.g. Coy et al. [13]) and self-adaptive procedures (e.g. Kivijärvi et al. [19]).

In this paper we show that parallel implementations of metaheuristics based on simple parallel models are efficient tools for both experimenting and fine tuning of parameters. Our proposal is based on two parallel models: the Independent Runs (IR) and the Independent Runs with Autonomous Strategies (IRAS). To illustrate our proposal, we implement the Tabu Search for the 0-1 Multidimensional Knapsack problem in both models. We use the two implementations for experimenting and fine tuning of parameters obtaining high quality solutions as compared with best known up-to-date results for the problem.

1 Introduction

Metaheuristics were introduced in the last two decades as a new kind of approximate algorithms which combine heuristic methods with higher level frameworks (see the survey in [7]). A good definition is the one provided in the Metaheuristics Network website [20]:

"A metaheuristic is a set of concepts that can be used to define heuristics methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework which can be applied to different optimization problems with relatively few modifications to make them adapted to a specific problem".

The implementation of metaheuristics for combinatorial optimization problems is, in general, a complex task. This can be easily seen if we consider that a metaheuristic involves three other main concepts:

\[
\text{Metaheuristic} = \text{Main method (framework)} + \text{internal/external heuristics} + \text{setting of search parameters}.
\]

* Partially supported by the CICYT Project TIC2002-04498-C06-03 (TRACER). M. Blesa acknowledges support by the Catalan Research Council of the Generalitat de Catalunya (grant no. 2001FI-00669).
For example, the Tabu Search metaheuristic [15,16] consists of the main method, several internal heuristics such as intensification, diversification and other external methods/modules, for example a module for managing the tabu list. Clearly, several parameters such as the tabu list size, the number of intensifications/diversifications etc. must be set to appropriate numerical values for the good performance and success of the method. From the high level decomposition of a metaheuristic we see that an efficient implementation of it requires a considerable effort. As a matter of fact, usually implementing a metaheuristic for a combinatorial optimization problem is reduced to implementing the main method itself and other internal/external heuristics used in it. We naturally can ask: are we done? Clearly not! Having completed the implementation does not mean that we are done since we still need a right setting of search parameters in order for our implementation to be successful.

The setting of the parameters, though not considered as part of the implementation, may well result even much more difficult than the implementation itself. On the one hand, this is shown to be true by practitioners’ experience where in many cases the time needed to find a right setting of parameters exceeds the time needed for the implementation (sometimes the time needed for fine tuning of parameters can be disparate.) On the other hand, the implementation effort is nowadays somewhat “simplified” due to sort of standardization of the main method and/or internal heuristics as well as due to the existence of several efficient libraries of data structures and algorithms that alleviate the development effort. Indeed, we can find now in literature templates for most known metaheuristics (e.g. Tabu Search template in [17], Memetic Algorithms template in [21]) that are generic enough to encompass most implementations which can be found in the literature. These templates can serve as a basis for providing generic implementations thus, reducing again the development time for metaheuristic implementation thanks to reusability.

So, while implementation issues might be considered well understood and to certain extent not time consuming, the setting of parameters still remains the backbone of metaheuristic implementations. As we will mention later, several approaches have been proposed to deal with the experimenting and fine tuning of parameters for metaheuristics, however, they partially succeed to solve the problem. Indeed there is still a lot of room left here, as pointed out by Barr et al. [5]

"...the selection of parameter values that drive heuristics is itself a scientific endeavor and deserves more attention than it has received in the operations research literature."

In this paper we address the issue of using parallel implementations as a means for efficient experimenting and fine tuning of parameters for metaheuristics. We show that parallel implementations of metaheuristics based on simple parallel models are efficient tools for both experimenting and parameters’ tuning. Our proposal is based on two parallel models: the Independent Runs (IR) and the Independent Runs with Autonomous Strategies (IRAS). As we will see, these parallel programs can be obtained easily from the sequential implementation of the metaheuristic, therefore, little extra-effort is needed for them. To illustrate our proposal, we consider the Tabu Search implementation for the 0-1 Multidimensional Knapsack problem and show how to obtain simple parallel programs in the both models mentioned above. Then we use the two parallel implementations of the methods for experimenting and fine tuning of the parameters involved with the Tabu Search method and the problem. We obtain high quality solutions when running the program with the chosen parameters, as compared with best known up-to-date results for the problem.

The paper is organized as follows. We review previous work on this topic in Section 2. The two parallel models we proposed are explained in Section 3. In Section 4 we give further details on how these two parallel models are designed and implemented. We give in Section 5 a case study of the Tabu Search metaheuristic for the 0-1 Multidimensional Knapsack problem. In the same section, we introduce the parameters involved and detail their tuning, together with some experimental results obtained with the setup determined by tuned values. We give some conclusions and point out further work in Section 6.
2 Previous work

The experimenting and fine tuning of parameters have deserved the attention of researchers to deal with this tedious process. Two directions of research can be pointed out with this respect. On the one hand, researchers have tried to provide a methodology and rigorous basis for experimental evaluation of heuristics, e.g., Rardin and Uzsoy [22], Barr et al. [5]. This line of research tried to offer a kind of methodological framework to deal with the experimental evaluations of heuristics. Important aspects of the evaluation are the experimental design, finding good sources of test instances, measuring the algorithmic performance, analysis and presentation of results. As pointed out by several researchers, although the experimenting issues are in spirit similar to those of optimization algorithms in general, the experimental evaluation has its distinctive aspects that make it a challenge. Due to the great difficulty, the main issues of the experimental evaluation are just highlighted as guidelines for designing experiments to test heuristic methods and for reporting on the experimentation results.

There is, however, one issue not taken into consideration: the time needed to complete the experimenting. The time issue is very relevant when a large benchmark of instances is going to be used in testing the metaheuristic. Running the implementation once on an instance may lead to reporting insignificant results since in most cases metaheuristic implementations are non-deterministic. Hence, one have to run the implementation on the same instance and the same settings of parameters several times and report the average cost of best encountered solutions. This is known in the literature as multiple-run single-instance (e.g., [12]) – in the deterministic implementation we have single-run single-instance. It’s obvious that multiple-run requires a considerable amount of time. The time issue has even more impact in case of benchmarks in which we have group of instances of the same characteristics rather than a single instance. In such a case we have multiple-run multiple-instance, that is, we have to report an average cost for the instance group requiring, thus, for any instance of the group a multiple-run single-instance, and we have to do this for all the groups of the benchmark.

On the other hand, different ways to deal in practice with the complex task of fine tuning of parameters have been proposed. Those different approaches focus on different important aspects, like the development of specific software (apropos of computing accurate values for search parameters of a given heuristic, e.g., the CALIBRA procedure [1]), the use of experimental design (on purpose of effective parameter settings based on statistics and gradient descent methods [13]), the use of other heuristics (by using Genetic Algorithms to Fine-Tune Instruction-Scheduling Heuristics), and the study of self-adaptive procedures [19].

3 Parallel models for experimenting and fine tuning of parameters

Usually the parallelism, through different parallel models, has been used to reduce the computation time. For our purpose we describe here two simple parallel models: the Independent Runs (IR) and the Independent Runs with Autonomous Strategies (IRAS) (the reader is referred to [9,11,10] for a complete taxonomy of parallel models for metaheuristics.) These two models are closely related (the IR model can be seen as a special case of IRAS model), however, we describe them separately since they are used here with different objectives: the IR model is intended for experimenting while IRAS model for the fine tuning of parameters.

3.1 Independent Runs model (IR)

The Independent Runs model, also known in literature as p-control + rigid synchronization consists of simultaneous and independent executions of the same program. In this model, there is a processor doing the coordinator task that consists in, at the beginning, sending the problem instance as well as the numerical values for the parameters to the rest of processors and, at the end, receiving the results upon termination of all the processor execution (see Figure 1). In this model each processor runs the same instance of the program and the communication time is almost non-existent. Observe that this model make sense as far as the program is non-deterministic.
IV

This is precisely the case of metaheuristic implementations which take randomized or probabilistic decisions. Therefore, running the same implementation in different processors leads to exploring different areas of the search space via different paths.

In general, running the parallel IR implementation on $p$-processors is equivalent to running the program $p$-times sequentially. Clearly, there is no reason to expect that the IR model yields better results than the sequential program, though in practice it tends to give better results. There is, however, a clear gain in time: if we run the sequential program for $t$ steps of time, in parallel we can run the same program $k$ times ($k \leq p$) within the same time $t$. In most of the cases we can easily achieve $k \approx p$ since the overhead due to the parallelism (distributing the input and recollecting the results) is very small. Similarly, if the machines where the parallel execution takes place are homogenous then, the execution time of any processor is almost the same since the work load is the same. Obviously, in heterogenous machines the execution time will be determined by the slowest machine.

Thus, in general, we can scale down the experimentation time with a factor of up to $p$ –the number of processors. For this reason, this model is then very suitable for experimental evaluation of metaheuristic implementations, especially the multiple-run single-instance and multiple-run multiple-instance cases.

This model has been usually used for LAN (Local Area Network) environments where many commodity machines are available. Nowadays, this model is even more attractive since it is quite suitable for implementations in WAN (Wide Area Network) environments due to the scarce number of communications between the processors. Since in WAN environments we can dispose of a high number of commodity machines, we can significantly scale down the experimentation time and to report them more trustworthy results.

3.2 Independent Runs with Autonomous Strategies (IRAS)

This model is quite similar to the Independent Runs. The IR model can be seen as a special case of the IRAS model in which the processors are given, additionally, a strategy to be used for its own search. A strategy is defined as an $m$-tuple

$$\text{strategy} = (\text{parameter}_1, \text{parameter}_2, \ldots, \text{parameter}_m),$$

where each parameter$_i$ is a different parameter of the metaheuristic. For each processor proc$_i$, the coordinator processor proc computes a strategy $S_i$, and then sends it together with the problem instance to the processor proc$_i$ (see Figure 1).

Clearly, using this parallel model we can efficiently make the fine tuning of parameters. While by a sequential execution we would try only one setting of metaheuristic parameters, in the IRAS model we can try $p$ -the number of processors- different settings of parameters within almost the same time as in sequential execution and compute the best among them.

4 Generic parallel implementations based on IR and IRAS models

The implementation of both parallel models can be fully generic and independent of the (sequential) metaheuristic implementation at hand. To this end we design the simple class diagram in Figure 2, in which the class SolverJAN will be in charge of running the parallel program while SolverJR and SolverIRAS (that extend the SolverJAN via inheritance) will implement specifically the task of coordinator processor (aka “master”) as well as that of slave processor. SolverSeq denotes the sequential implementation of the metaheuristic, through which we can declare an instance of such implementation and run the main method. The one-to-one relationship between SolverJAN and SolverSeq shows that SolverJAN will use/declare instances of SolverSeq used as a black-box by the parallel program. Observe also that Instance, Setup, Strategy and Solution are entities for representing the problem instance data, parameters, strategy and a feasible solution to the problem, respectively. Those entities are problem-dependent and
will be implemented according to the problem at hand. The parallel program provides their interfaces and hence can use them as black-boxes.

The implementation is done in C++ language using the MPI (Message Passing Interface) as a communication library [18]. The entities appearing in the class diagram of Figure 2 are translated to C++ classes (for example, see Figure 3 for the interface of Solver.lan and its method run(), in which the methods master() and slave() are common to both derived classes Solver.IR and Solver.IRAS and are implemented in each class accordingly.)

Running the parallel program. The classes Solver.IR and Solver.IRAS implementing the IR and the IRAS models, respectively, are generic. Therefore, they just need to be implemented once, but can be further reused to run in parallel as many (sequential) implementations of combinatorial optimization problems as desired. This is an important basis towards a generic parallel framework for metaheuristics (see [2, 3]). The program is run through the Solver.IR or Solver.IRAS class after creating an object of it (see Figure 4).

The run() method executes the corresponding parallel model. In its implementation it uses abstract classes representing the main concepts of the metaheuristic method, e.g., the solution, see the class diagram in Figure 2. This generic way of designing and implementing the framework has important benefits, like genericity and reusability.

5 Case study: Tabu Search for the 0-1 Multidimensional Knapsack problem

To illustrate our proposal, we consider the Tabu Search implementation for the 0-1 Multidimensional Knapsack problem. Two solvers, a Solver.IR and a Solver.IRAS as explained in the previous sections, are created for the use of a generic sequential implementation of the Tabu Search method. We use the two parallel implementations for experimenting and for the fine tuning of the parameters involved with the Tabu Search method and the problem.

Tabu Search (TS) [15,16] belongs to the family of local search algorithms but here the search is done in a guided way in order to overcome the local optima. Roughly speaking, the method starts from an initial solution and jumps from one solution to another one in the solution space.
class Solver_LAN
{
    public:
    Solver_LAN (const Problem& probl,
                 const Setup& setup);
    "Solver_LAN ()
    void run ()
    protected:
    const Problem& _instance;
    const Setup& _setup;
    Solution _best_solution;
    int _time_spent;
    private:
    int _comm_size;
    void master();
    void slave();
};

void Solver_LAN::run ()
{
    // Start parallel execution environment
    int err, rank;
    MPI_Comm_size(MPI_COMM_WORLD, &comm_size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if(rank==0)
        master();
    else
        slave();
}

#include 'Metaheuristic.hh'
int main (int argc, char** argv)
{
    Problem& instance = ... // Read the instance
    Setup& setup = ... // Read the setup parameters
    // Declare and execute an instance of parallel solver.
    Solver_LAN solver(instance, setup);
    solver.run();
    // Report the processor number which found to best solution.
    cout << solver.best_cost() << endl;
    cout << solver.time_spent() << endl;
    // The processor with this number used the best strategy.
    cout << solver.pid() << endl;
    return 0;
}

Fig. 3. Class Solver_LAN in the header file Metaheuristic.hh (left) and its method run() (right)

Fig. 4. How to run the IRAS model. An object of the class Solver_IRAS is created from the problem instance and the setup parameters. The program is executed with the method run(). Once the execution is finished, different information of the search process can be accessed, e.g., the best solution found or the time required to find it.
but tries to avoid cycling by forbidding or penalizing moves which take the solution, in the next iteration, to solutions previously visited (called *tabu*). To this aim, TS keeps a *tabu list* which is historical in nature and constitutes the tabu search memory. The role of the memory can change as the algorithm proceeds. At initialization the goal is to make a coarse examination of the solution space and further on the search is focused to produce local optima solutions in a process of *intensification* or make a *diversification* in order to explore new regions of the solution space.

The *NP-hard 0-1 Multidimensional Knapsack problem* (0-1MKNP) consists in selecting a subset of *n* given objects in such a way that the total profit of the selected objects is maximized while a set of knapsack constraints are satisfied. The 0-1MKNP problem can be stated as:

\[
\text{maximize } c \cdot x \\
\text{subject to: } Ax \leq b, \quad x \in \{0,1\}^n
\]

where \(c \in \mathbb{N}^n\), \(A \in \mathbb{N}^{m \times n}\), and \(b \in \mathbb{N}^m\). The binary components \(x_j\) of \(x\) are decision variables: \(x_j = 1\) if the object \(j\) is selected, and \(x_j = 0\) otherwise. The profit associated to \(j\) is denoted by \(c_j\). Each of the \(m\) constraints \(A_i x \leq b_i\) is called a knapsack (capacity) constraint.

### 5.1 Parameters involved

Many parameters are involved in an instantiation of the 0-1 Multidimensional Knapsack problem via Tabu Search. Some of these parameters are due to the problem while some others are proper to the metaheuristic method. Usually, the former are fixed in every instance of the problem, but most of the latter require a tuning process.

Five parameters define the 0-1MKNP: the number of variables (objects) \(n\), the number of constraints \(m\), the profits of the \(n\) objects \(c \in \mathbb{N}^n\), the matrix of constraints \(A \in \mathbb{N}^{m \times n}\), and the capacities of the knapsack \(b \in \mathbb{N}^m\) at each of the dimensions. Every fixed set of values for these parameters defines an instance of the problem and, according to them, instances can be easier or harder to solve. This is an important feature to consider when studying the robustness and the performance of an algorithm.

Many parameters control the Tabu Search method (see Table 1). The basic parameters are those controlling the stopping conditions of the algorithm (max_execution_time and independent_runs) and the influence of the historical search memory (tabu_list_size). Other important parameters are used to control the search process, specially the neighborhood exploration (max_neighbors), the intensification (max_repetitions, nb_best_sols and nb_intensifications) and the diversification (history_rep and nb_diversifications).

### 5.2 Fine tuning

The performance of the Tabu Search algorithm when applied to any combinatorial optimization problem depends strongly on the values given to the parameters introduced above. Moreover, those parameters are mutually and strongly dependent. For the success of the method, those parameters have to be tuned in order to find appropriate values. We have tuned the Tabu Search parameters by using the IRAS model introduced in Section 3.2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_execution_time</td>
<td>Maximum execution time of each independent run of the algorithm.</td>
</tr>
<tr>
<td>independent_runs</td>
<td>Number of independent runs.</td>
</tr>
<tr>
<td>tabu_list_size</td>
<td>Size of the tabu list (for the short term memory of the TS method).</td>
</tr>
<tr>
<td>max_neighbors</td>
<td>Maximum number of neighbors to explore.</td>
</tr>
<tr>
<td>max_repetitions</td>
<td>Number of non-improvements before the intensification is triggered.</td>
</tr>
<tr>
<td>nb_best_sols</td>
<td>Number of best solutions maintained as history for the intensification.</td>
</tr>
<tr>
<td>nb_intensifications</td>
<td>Number of iterations in which the search is intensified.</td>
</tr>
<tr>
<td>history_rep</td>
<td>How often objects can belong to a solution before the diversification.</td>
</tr>
<tr>
<td>nb_diversifications</td>
<td>Number of iterations in which the search is diversified.</td>
</tr>
</tbody>
</table>
Fig. 5. Details related to the \texttt{max\_repetitions}, \texttt{nb\_intensifications} and the \texttt{history\_rep} parameters.

In the following, we report the values obtained for each of the Tabu Search parameters in Table 1 after the fine tuning done via the IRAS model.

\textbf{max\_execution\_time and independent\_runs}. We found that 2 independent runs of the program already provided relatively good solutions, but depending strongly on the number of iterations of each independent run. The number of iterations required depend on the size of the problem, specially on \( n \) (the number of variables). We found that, for small and middle-sized instances 100n iterations are enough to obtain good results, while for big instances (with more than 100 variables) 1000n iterations are needed.

\textbf{tabu\_list\_size}. We have found that, for the instances we have tried, a tabu list size of length in \([3, \ldots, 15]\) is good. The final value is chosen randomly in this interval.

\textbf{max\_repetitions}. Not specific value could be decided for this parameter from our experiments. Therefore, we decided to explore the whole neighborhood instead of only a portion of it.

\textbf{max\_repetitions}. It is important to choose an appropriate value for this parameter, since it makes the intensification process to start. A too-low value for this parameter can make the intensification to start before it is really needed, which might represent yet a too strong search process. A too-high value for this parameter delays the intensification, thus producing a waste of time meanwhile. Since it is very strict and unlikely that exactly the same solution is obtained in consecutive iterations, we have relaxed the definition of difference between solutions by considering the Hamming distance between them. Then, we define that two solutions are considered the same if they have: (a) at most \( an \) \((0 < \alpha \leq 1)\) and, (b) at most \( an/\ln n \) \((0 < \alpha \leq 1)\) differences. In the first case, good results are obtained only for small instances and values of \( \alpha \) around 0.05; however, the number of differences grows too fast when bigger instances are considered. In the second case, the smoother behavior of the expression leads to better results (see Figure 5(a)) and experiments obtain good results for \( \alpha = 0.1 \).

\textbf{nb\_best\_sols}. The bigger the values for this parameter, the more historical information available for a good intensification. However, this also makes the intensification processes computationally more expensive. The experiments performed suggest that keeping between 10 and 15 solutions is a good equilibrium between the benefit of the intensification and its computational cost.

\textbf{nb\_intensifications}. We have experimentally observed in the tuning process that even small values of this parameter are useful, thus justifying the need for intensification in the search process. Again we had to find a good trade-off between the computational cost and the contribution of the intensification to the quality of solutions (success of the intensification). According to the experiments (see Figure 5(b)), we observed that a high percentage of success for intensification can be obtained for the two value of parameter close to 10.

\textbf{history\_rep}. The better results are obtained for high values of this parameter (between 80% and 95%, see Figure 5(c)), which will make the diversification to be a smooth process. However, in order to assure a good long-term influence of this parameter we use internally an adaptive relative expression of it.
5.3 Computational results

After tuning the parameters involved in the Tabu Search method for the 0-1 Multidimensional Knapsack problem, we have tested the program more in detail with values provided by the tuning. In order to obtain some statistical significance about the robustness of the algorithm, the same instance should be run several times with the same parameters setting and the results should refer to the average results obtained. To fasten this process we use the IR parallel model proposed in Section 3.1, which we run in a cluster of computers AMD k6-11 with 450 MHz processors and 256Mb of memory.

We test small, medium and big instances (in terms of the number of variables) taken from the literature. Small instances ($n \leq 50$) are taken from [14, 8], middle-sized instances ($50 < n \leq 100$) are taken from [24, 23], and big instances ($100 < n \leq 500$) are taken from the OR-library [6]. Since our aim is to test how does our generic implementation and parallel fine tuning of parameters behave, we have chosen instances for which the optimum value is known and, thus, we could establish a comparison with the results obtained from our approach. Note that the optimum values are obtained through exact methods, e.g., Branch and Bound, which require enormous computational resources. The main advantage of metaheuristics contrast to that, since they can obtain relatively good solutions in a (sometimes very) short time. Table 2 shows the results we obtained, where we can observe that the deviation of the cost of our solutions from the optimum is very low. On the one hand this shows that the values of the parameters that we found through our approach are appropriate and, on the other, the low values for the deviation show the robustness of our approach in the sense that the values we found for the parameters perform very well for a large set of different instances.

<table>
<thead>
<tr>
<th>Instance</th>
<th>n</th>
<th>m</th>
<th>Optimum</th>
<th>Best cost</th>
<th>Average cost</th>
<th>deviation</th>
<th>Iterations</th>
<th>time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNAP15</td>
<td>15</td>
<td>10</td>
<td>4063</td>
<td>4063</td>
<td>4063</td>
<td>0</td>
<td>3600</td>
<td>3.7</td>
</tr>
<tr>
<td>KNAP20</td>
<td>20</td>
<td>10</td>
<td>6120</td>
<td>6120</td>
<td>6120</td>
<td>0</td>
<td>1600</td>
<td>2.5</td>
</tr>
<tr>
<td>KNAP25</td>
<td>25</td>
<td>10</td>
<td>8196</td>
<td>8196</td>
<td>8196</td>
<td>0</td>
<td>2000</td>
<td>4.1</td>
</tr>
<tr>
<td>KNAP30</td>
<td>30</td>
<td>10</td>
<td>10618</td>
<td>10618</td>
<td>10618</td>
<td>0</td>
<td>1000</td>
<td>16.8</td>
</tr>
<tr>
<td>KNAP40</td>
<td>40</td>
<td>15</td>
<td>11537</td>
<td>11537</td>
<td>11537</td>
<td>0</td>
<td>1000</td>
<td>21.3</td>
</tr>
<tr>
<td>SENTO1</td>
<td>60</td>
<td>30</td>
<td>772</td>
<td>772</td>
<td>772</td>
<td>0</td>
<td>5000</td>
<td>63.3</td>
</tr>
<tr>
<td>SENTO2</td>
<td>60</td>
<td>30</td>
<td>872</td>
<td>872</td>
<td>872</td>
<td>0</td>
<td>5000</td>
<td>55.5</td>
</tr>
<tr>
<td>WEINT15</td>
<td>15</td>
<td>2</td>
<td>1068445</td>
<td>1068445</td>
<td>1068445</td>
<td>0</td>
<td>4000</td>
<td>18.7</td>
</tr>
<tr>
<td>WEINT20</td>
<td>20</td>
<td>2</td>
<td>1068445</td>
<td>1068445</td>
<td>1068445</td>
<td>0</td>
<td>4000</td>
<td>18.7</td>
</tr>
<tr>
<td>OR10x100</td>
<td>100</td>
<td>10</td>
<td>23064</td>
<td>23064</td>
<td>23064</td>
<td>0.0254</td>
<td>5029</td>
<td>600</td>
</tr>
<tr>
<td>OR10x250</td>
<td>250</td>
<td>10</td>
<td>59187</td>
<td>59187</td>
<td>59187</td>
<td>0.0502</td>
<td>5132</td>
<td>900</td>
</tr>
<tr>
<td>OR250x500</td>
<td>500</td>
<td>10</td>
<td>117726</td>
<td>117726</td>
<td>117726</td>
<td>0.0506</td>
<td>35487</td>
<td>1200</td>
</tr>
<tr>
<td>OR300x100</td>
<td>100</td>
<td>30</td>
<td>21946</td>
<td>21946</td>
<td>21946</td>
<td>0.0151</td>
<td>31015</td>
<td>600</td>
</tr>
<tr>
<td>OR300x250</td>
<td>250</td>
<td>30</td>
<td>56993</td>
<td>56993</td>
<td>56993</td>
<td>0.0350</td>
<td>15215</td>
<td>900</td>
</tr>
<tr>
<td>OR300x500</td>
<td>500</td>
<td>30</td>
<td>115806</td>
<td>115806</td>
<td>115806</td>
<td>0.0307</td>
<td>10459</td>
<td>1200</td>
</tr>
</tbody>
</table>

6 Conclusions and further work

We have shown that the parallelism is a useful mean to deal with the tedious and time consuming process of experimenting and fine tuning of parameters of metaheuristics implementations. Interestingly, this can be done through simple parallel models, namely, the Independent Runs and the
Independent Runs with Strategies. We have applied our approach in experimenting and fine tuning of parameters of Tabu Search for 0-1 Multidimensional Knapsack problem and the computational results validate the approach.

We would like to fully automatize the process of obtaining the parallel programs from any existing sequential implementation of metaheuristics and make the use of the parallel programs more user-friendly.

References

Department de Lenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Research Reports - 2003

- LSI-03-1-R: Metaheuristics for the Edge-Weighted $g$-Cardinality Tree Problem, Blesa, M.J. and Blum, C.
- LSI-03-3-R: A Complete Solid Model for Surface Rendering, Aguilera, A. and Ayala, D. and Rodriguez, J.
- LSI-03-4-R: Fast Neighborhood Operations for Images and Volume Datasets, Ayala, D. and Rodriguez, J.
- LSI-03-5-R: A Review of Integrity Constraint Maintenance and View Updating Techniques, Mayol, E. and Teniente, E.
- LSI-03-6-R: Porgpine: A Peer-to-Peer Search Engine, Bermúdez, J. and Pujol, J.M. and Sangüesa, R.
- LSI-03-7-R: KDSM Methodology for knowledge discovery from a domain where repeated very short serial measures with a blocking factor are present, Alvarado, G. and Cortés, U. and Gibert, K. and Rodas, J. and Rojo, J.E.
- LSI-03-8-R: On the Maximum Common Embedded Subtree Problem for Ordered Trees, Valiente, G.
- LSI-03-9-R: Searching by approximate personal-name matching, Camps, R. and Daudé, J.
- LSI-03-10-R: Two triangulations methods based on edge refinement, Ayala, D. and Pla, N. and Vigo, M.
- LSI-03-11-R: Constrained Tree Inclusion, Valiente, G.
- LSI-03-12-R: Geometry of language and linguistic circuitry, Morrill, G.
- LSI-03-13-R: A simple yet useful approach to implementing UML Profiles in CASE tools ExtendedVersion, Cabot, J. and Gómez, C.
- LSI-03-14-R: Solving 'Still life' with Soft Constraints and Bucket Elimination, Larrosa, J. and Morancho, E.
- LSI-03-15-R: A fast hierarchical traversal strategy for multimodal visualization, Puig, A. and Tost, D.
- LSI-03-16-R: The complexity of deciding stability under FFS in the adversarial model, Álvarez, C. and Blesa, M.J. and Díaz, J. and Fernández, A. and Serna, M.
- LSI-03-17-R: A Case Study on Building Conceptual Schemas by Refining General Ontologies, Conesa, J. and de Palol, X.
- LSI-03-19-R: Análisis de medidas repetidas mediante el uso de la curva media como síntesis,
• LSI-03-20-R : Statistical Strategies for Pruning All the Uninteresting Association Rules, Casas, G.
• LSI-03-21-R : Discovering Unbounded Episodes in Sequential Data, Casas, G.
• LSI-03-22-R : Query Containment with Negated IDB Predicates Extended Version, Farré, C. and Teniente, E. and Urpi, T.
• LSI-03-24-R : Using concept lattices to mine functional dependencies, Baixeries, J.
• LSI-03-25-R : Adversarial models for priority-based networks, Álvarez, C. and Blesa, M.J. and Fernández, A. and Serna, M.
• LSI-03-26-R : Characterization of Concept Lattices for Ordered Contexts, Casas, G.
• LSI-03-27-R : Computation of bisection width for random d-regular graphs, Serna, M. and Serras, J. and Wormald, N.C.
• LSI-03-28-R : Building and Using Quality Models for Complex Software Domains, Carvallo, J.P. and Franch, X. and Quer, C.
• LSI-03-30-R : Margin Maximization with Feed-forward Neural Networks: A Comparative Study with Support Vector Machines and AdaBoost, Carreras, X. and Márquez, Ll. and Romero, E.
• LSI-03-31-R : Una notación algorítmica con genericidad y herencia junto con su relación con C++ y Java, Mylonakis, N.
• LSI-03-32-R : Study on k-Shortest Paths with Behavioral Impedance Domain from the Intermodal Public Transportation System Perspective, Madriz-Lozada, E. and Pereira, H. and Perez, Ll.
• LSI-03-33-R : On the Complexity of Resolution with bounded Conjunctions, Borrelleras, C. and Esteban, J.L. and Messner, J.
• LSI-03-34-R : Generic Algorithms for the Generation of Combinatorial Objects, Martínez, C. and Molinero, X.
• LSI-03-35-R : An Efficient Generic Algorithm for the Generation of Unlabelled Cycles, Martínez, C. and Molinero, X.
• LSI-03-36-R : CBR and MBR techniques: review for an application in the emergencies domain, Mérida-Campos, C. and Rollón, E.
• LSI-03-37-R : Using NLP tools in the Specification Phase, Castell, N. and Hernández, A.
• LSI-03-38-R : Classificació automàtica amb KLASS de les dades de procés d’una EDAR, Flores, X. and Gibert, K. and Rodríguez-Roda, I.
• LSI-03-39-R : VolumeEVM: A new surface/volume integrated model, Ayala, D. and Rodríguez, J.
• LSI-03-40-R : MKtrees: Construction and Applications, Brunet, P. and Franquesa, M.
• LSI-03-41-R : From Supply Chains to Demand Networks: Agents in Retailing: The Electrical Bazaar, Almirall, E. and Brito, I. and Cortés, U. and Silisque, A.
\begin{itemize}
\item LSI-03-42-R: \textit{Fixed-Parameter Algorithms for the \(k, r\)-Center in Planar Graphs and Map Graphs}, Demaine, E.D. and Fomin, F.V. and Hajiaghayi, M. and Thilikos, D.M.
\item LSI-03-43-R: \textit{Dominating sets and local treewidth}, Fomin, F.V. and Thilikos, D.M.
\item LSI-03-44-R: \textit{Res2 Lower Bounds for Tseitin Formulas}, Bonet, M. and Esteban, J.L.
\item LSI-03-45-R: \textit{Collision Queries: Models and Algorithms}, Brunet, P. and Franquesa, M.
\item LSI-03-46-R: \textit{Collision Prediction Using MKtrees: Broad Phase and Refinement Levels of the Narrow Phase}, Brunet, P. and Franquesa, M.
\item LSI-03-48-R: \textit{Bidimensional Parameters and Local Treewidth}, Demaine, E.D. and Fomin, F.V. and Hajiaghayi, M. and Thilikos, D.M.
\item LSI-03-50-R: \textit{A distributed algorithm to find Hamiltonian cycles in Gnp random graphs}, Levy, J. and Louchard, G. and Petit, J.
\item LSI-03-51-R: \textit{A characterization of universal stability in the adversarial queuing model}, Álvarez, C. and Blesa, M.J. and Serna, M.
\item LSI-03-52-R: \textit{Automatic Generation of Polynomial Loop Invariants for Imperative Programs}, Kapur, D. and Rodríguez, E.
\item LSI-03-53-R: \textit{Speeding up rendering of hybrid surface and volume models}, Ferré, M. and Puig, A. and Tost, D.
\item LSI-03-54-R: \textit{A Simple and Fast Approach for Solving Problems on Planar Graphs}, Fomin, F.V. and Thilikos, D.M.
\item LSI-03-55-R: \textit{Roles as Entity Types: A Conceptual Modelling Pattern}, Cabot, J. and Raventós, R.
\item LSI-03-56-R: \textit{Using Parallelism in Experimenting and Fine Tuning of Parameters for Metaheuristics}, Blesa, M.J. and Xhafa, F.
\end{itemize}

Hardcopies of reports can be ordered from:

Núria Sanchez  
Departament de Lenguatges i Sistemes Informàtics  
Universitat Politècnica de Catalunya  
Campus Nord, Mòdul C6  
Jordi Girona Salgado, 1-3  
03034 Barcelona, Spain  
nurias@lsi.upc.es

See also the Departament WWW pages, http://www.lsi.upc.es/