Using Parallelism in Experimenting and Fine Tuning of Parameters for Metaheuristics*

Maria Blesa and Fatos Xhafa

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

Abstract. We show that parallel implementations of metaheuristics are efficient tools for both experimenting and fine tuning of parameters.

1 Introduction

Metaheuristics were introduced in the last two decades as a new kind of approximate algorithms for solving combinatorial optimization problems which combine heuristic methods with higher level frameworks [5]. Their implementation is usually a complex task, since they involve three main concepts: (1) *Main method*, (2) *internal/external heuristics*, and (3) *setting of search parameters*.

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. Moreover, finding of right values for the search parameters of the metaheuristic is almost indispensable for the success of the metaheuristic implementation. 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 [11,2] and, on the other, to find efficient approaches for fine tuning of parameters such as developing specific software [1], use of experimental design [7] and self-adaptive procedures [10].

We address the issue of using parallel implementations as a mean for efficient experimenting and fine tuning of parameters for metaheuristics. Our proposal is based on two parallel models and, to illustrate our proposal, we have applied it in experimenting and fine tuning of parameters for the Tabu Search method applied to the 0-1 Multidimensional Knapsack problem. High quality solutions as compared with best known up-to-date results for the problem are obtained.

2 Parallel Models for Experimenting and Fine Tuning

Parallelism has been usually used to reduce computation times. For our purpose we describe here two simple parallel models: the *Independent Runs* (IR) and

* Partially supported by the CICYT Project TIC2002-04498-C05-03 (TRACER) and by the Catalan Research Council of the Generalitat de Catalunya (grant no. 2001FI-00659). *For a longer version of this work, see [4].*

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3036, pp. 429–432, 2004.

the *Independent Runs with Autonomous Strategies* (IRAS). Although closely related, they are used here with different objectives: the IR model is intended for experimenting while IRAS model for the fine tuning of parameters.

In the **Independent Runs model (IR)** there is a coordinator processor sending the problem instance and parameters' setup and receiving the results, and each processor runs the same instance of the program. Observe that this model make sense as far as the program is non-deterministic. This is precisely the case of metaheuristic implementations which take randomized or probabilistic decisions. Running the same implementation in p different processors leads to exploring different areas of the search space and it is equivalent to performing p sequential executions, thus scaling down the experimentation time with a factor of up to p. For this reason, this model is then very suitable for experimental evaluation of metaheuristic implementations. The **Independent Runs with Autonomous Strategies (IRAS)** can be seen as a special case of the IR 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 $\langle parameter_1, \ldots, parameter_m \rangle$, where each $parameter_i$ is a different parameter of the metaheuristic. For each processor $proc_i$, the *coordinator processor* $proc_0$ computes a strategy S_i, and then sends it together with the problem instance to the processor $proc_i$. Clearly, using this parallel model we can efficiently make the fine tuning of parameters.

Our implementation of both parallel models is fully generic and independent of the (sequential) metaheuristic implementation at hand. This is achieved through a careful class design and implementation in C++ using the MPI as a communication library. The class `Solver_LAN` will be be in charge of running the parallel program while the sub-classes `Solver_IR` and `Solver_IRAS` will implement specifically the task of coordinator and slave processors. `Solver_Seq` denotes the sequential implementation of the metaheuristic, through which we can declare an instance of such implementation and run the main method. There is a one-to-one relationship between `Solver_LAN` and `Solver_Seq` since the former will use instances of the latter as a black-box. The classes `Instance`, `Setup`, `Strategy` and `Solution` represent 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. Any of the models is run via the method `run()` of the corresponding class `Solver_IR` or `Solver_IRAS`. This generic way of designing and implementing the framework has important benefits, like genericity and reusability. 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.

3 Tabu Search for the 0-1 Multidimensional Knapsack

To illustrate our proposal, we implement the Tabu Search (TS) for the 0-1 Multidimensional Knapsack problem both in the IR model (which is intended for experimenting) and the IRAS model (which is intended for the fine tuning.)

Table 1. Numerical values for the parameters.

nb_iterations	independent_runs	tabu_list_size	max_neighbors
$100n$ (small and middle) $1000n$ (big size)	2	$[3 \dots 15]$	full exploration
nb_best_sols	**nb_intensifications**	**history_rep**	**nb_diversifications**
$[10 \dots 15]$	$\simeq 10$	$80 - 95\%$	$\simeq 10$

Table 2. Results for the 0-1MKNP. Best and average costs obtained over 20 executions, respectively. The 7th column is the deviation of the sample wrt. the average. The last two columns indicate the number of iterations performed and time spent on it.

Instance	n	m	Optimum	Best cost	Avg. cost	deviation	Iters.	time (s)
KNAP15	15	10	4015	4015	4014.1	0	3000	3.6
KNAP20	20	10	6120	6120	6120.0	0	1600	2.5
KNAP50	50	5	16537	16520	16441.0	0.001	10000	21.3
SENTO1	60	30	7772	7772	7772.0	0	5000	53.5
SENTO2	60	30	8722	8722	8720.6	0	5000	55.5
OR10x100-00	100	10	23064	22478	22360.4	0.025	85629	600
OR10x250-00	250	10	59187	56213	55945.6	0.050	55132	900
OR10x500-00	500	10	117726	111773	111486.7	0.051	35487	1200
OR30x100-00	100	30	21946	21614	21520.7	0.015	31615	600
OR30x250-00	250	30	56693	54711	54534.6	0.035	15215	900
OR30x500-00	500	30	115868	111272	110942.4	0.040	10459	1200

Tabu Search [9] 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. The search process 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 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: maximize $c \cdot x$, subject to: $Ax \leq b$, $x \in \{0,1\}^n$, 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 $A_i x \leq b_i$ is a capacity constraint.

Parameters involved, fine tuning and computational results. Five parameters define the 0-1MKNP: the number of objects n, the number of constraints m, the profits of the objects $c \in \mathbb{N}^n$, the matrix of constraints $A \in \mathbb{N}^{m \times n}$, and the capacities $b \in \mathbb{N}^m$. 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. The basic parameters controlling TS are

concerned with stopping conditions (`nb_iterations`, and `independent_runs`) and the influence of the historical search memory (`tabu_list_size`). Other parameters control the search process, specially the neighborhood exploration (`max_neighbors`), the diversification (`history_rep` and `nb_diversifications`), and the intensification (`nb_best_sols` and `nb_intensifications`). All those parameters are mutually and strongly dependent. For the success of the method, appropriate values for those parameters have to be find. We have tuned the Tabu Search parameters by using the IRAS model introduced above.

After tuning these parameters (see Table 1), we have run the 0-1MKNP implementation in a cluster of computers AMD K6-11 with 450 MHz processors and 256Mb of memory (see [4]). 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 average results should be provided. We test small ($n \leq 50$), middle-sized ($50 < n \leq 100$) and big instances ($100 < n \leq 500$) taken from the literature [8,6,3]. 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 (obtained through computationally expensive exact methods) is known (see Table 2). The low values on the deviation of the cost of our solutions from the optimum shows both that the values of the parameters that we found through our approach are appropriate, and also 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.

References

1. B. Adenso-Diaz and M. Laguna. (2002). Fine tuning of Algorithms Using Fractional Experimental Designs and Local Search. Submitted.
2. R.S. Barr, B.L. Golden, J. Kelly, W.R. Stewart, M.G.C. Resende. (2001) Designing and Reporting Computational Experiments with Heuristic Methods. *Journal of Heuristics*, 1(1):9–32.
3. J.E. Beasley. (1990). OR-Library: Distributing Test Problems by Electronic Mail. *Journal of the Operational Research Society*, 41:1069–1072.
4. M. Blesa and F. Xhafa. (2003). Using Parallelism in Experimenting and Fine Tuning of Parameters for Metaheuristics. Technical Report no. LSI-03-56-R, UPC.
5. C. Blum and A. Roli. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308.
6. C. Cotta and J.M. Troya. (1998). A Hybrid Genetic Algorithm for the 0-1 Multiple Knapsack Problem. In *Artificial Neural Nets and Genetic Algorithms*, chapter 3, pp. 251–255. Springer-Verlag.
7. S.P. Coy, B.L. Golden, G.C. Runer, E.A. Wasil. (2000). Using Experimental Design to Find Effective Parameter Settings for Heuristics *Journal of Heuristics*, 7:77–97.
8. A. Freville and G. Plateau. (1990). Hard 0-1 multiknapsack test problems for size reduction methods. *Investigation Operativa*, 1:251–270.
9. F. Glover and M. Laguna. (1997). *Tabu Search.* Kluwer Academic Publishers.
10. J. Kivijärvi, P. Fränti and O. Nevalainen. (2003). Self-Adaptive Genetic algorithm for Clustering. *Journal of Heuristics*, 9:113–129.
11. R.L. Rardin and R. Uzsoy. (2001). Experimental Evaluation of Heuristic Optimization Algorithms: A Tutorial. *Journal of Heuristics*, 7:261–304.