A Parallel Co-evolutionary Metaheuristic

Vincent Bachelet and El-Ghazali Talbi

Université des Sciences et Technologies de Lille
LIFL - UPRESA 8022 CNRS
Bâtiment M3 Cité Scientifique,
59655 Villeneuve d'Ascq CEDEX, France
{bachelet, talbi}@lifl.fr

Abstract. In order to show that the parallel co-evolution of different heuristic methods may lead to an efficient search strategy, we have hybridized three heuristic agents of complementary behaviours: A Tabu Search is used as the main search algorithm, a Genetic Algorithm is in charge with the diversification and a Kick Operator is applied to intensify the search. The three agents run simultaneously, they communicate and cooperate via an adaptive memory which contains a history of the search already done, focusing on high quality regions of the search space. This paper presents CO-SEARCH, the co-evolving heuristic we have designed, and its application on large scale instances of the quadratic assignment problem. The evaluations have been executed on large scale network of workstations via a parallel environment which supports fault tolerance and adaptive dynamic scheduling of tasks.

1 Introduction

Usually, the parallelism is considered as a mean to increase the computational power in running several tasks simultaneously. However, a completely different approach has been appeared recently. The objective is no more to speed up the processing, and the implementation may be sequential or parallel. The main interest is the co-evolution of activities. According to this approach, the metaheuristic we present in this paper combines the two advantages of the parallelism: the computational power and the co-evolution. Our objective is to design a robust and efficient search method involving the co-evolution of well known basic agents, having complementary behaviours.

In the design of a efficient heuristic search strategy, two opposite criteria have to be balanced: the exploration (diversification task) of the search space and the exploitation (intensification task) of the solutions that have already been found. Moreover, some algorithms are rather well fitted in diversification and others are rather good in intensification. Instead of trying to improve diversifying heuristics and intensifying ones, a common idea is to hybridize both approaches in order to make both methods compensate themselves. In order to design a well-balanced metaheuristic, we propose the co-evolution of a search agent (a local search), a diversifying agent and an intensifying agent. The three agents exchange information via a passive coordinator called the adaptive memory.

In the remainder of the paper, we present an instance of the model of the parallel co-evolutionary metaheuristic we have describe above. In section 2, a NP-hard problem, the quadratic assignment problem (QAP), which we use as a testbed for this study, is presented. In section 3, the idiosyncrasies of the co-evolutionary metaheuristic are described, the three agents and the adaptive memory are detailed without loss of generality with regard to the search problem to be tackled. Then, we give the specifications of the co-operative hybrid involved in the application on the QAP. In section 4, we present the experiments carried out on a parallel environment which supports the adaptive dynamic scheduling of tasks and fault tolerance. Finally, we comment the results obtained on standard large scale instances which reveals the interest in the co-evolution.

2 The Quadratic Assignment Problem

The Quadratic Assignment Problem (QAP) is used in this study for the co-operative hybrid to be evaluated. However, our study is more general and the co-evolutionary metaheuristic we present here is not especially designed for the QAP.

The QAP is a NP-hard problem and exact algorithms (such branch and bound) are restricted to small instances ($n < 25$). Therefore, many heuristics have been proposed in the literature: an extensive survey and recent developments can be found in [3].

The QAP can be stated as follows. Given the size of the instance n, a set of n objects $O = \{O_1, O_2, ..., O_n\}$, a set of n locations $L = \{L_1, L_2, ..., L_n\}$, a flow matrix c, where each element c_{ij} denotes a flow cost between the objects O_i and O_j, a distance matrix d, where each element d_{kl} denotes a distance between location L_k and L_l, find an object-location bijective mapping $U : O \longrightarrow L$, which minimises the function ψ:

$$\psi(U) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} . d_{U(i)U(j)}$$

To represent a solution of the QAP, we use the same representation for all agents which is based on a permutation of n integers: $u = (u_1, u_2, ..., u_n)$ where $u_i = U(i)$ denotes the location of the object O_i. The distance between two permutations can be defined with regards to the pair-exchange operator (swap) usually used for the QAP. The distance $dist(u, v)$ between u and v is the minimum number of swap operations needed to move from u to v.

3 CO-SEARCH: A parallel co-evolutionary metaheuristic

In the metaheuristic CO-SEARCH (CO-evolutionary Search), three heuristic agents run simultaneously and exchange information via an adaptive memory (AM). As the main search agent, we choose the tabu search (TS) because it is one of the most efficient local search methods now available in terms of solution quality. For the diversification task, we use a genetic algorithm (GA) as it is powerful at the exploration task due to its genetic operators. The intensification task is handled by a kick operator (KO) for it is a uncomplicated way to scramble, more or less, a solution.

3.1 MTS: the search agent

The search agent we propose is a multiple tabu search (MTS). The TS approach has been proposed by Glover [4]. Sometimes, The TS involves some techniques to encourage the exploitation of a region in the search space, or to encourage the exploration of new regions. For this study, the TS does not need any advanced technique for the diversification or the intensification because of the other agents.

We use multiple independent TS tasks running in parallel without any direct cooperation (between the different TS). We choose a multiple short TS run rather than a long run because the TS does not need to evolve far from the initial solution (in other regions). In CO-SEARCH, the TS has been extended with a frequency memory for storing an information about all the solutions it visits along a walk. Actually, the frequency memory stores the frequency of occurency of some events during the search [4]. Let H, the set of events to deal with, and let F_e, the frequency (number of occurency) of the event $e \in H$.

For solving the QAP, the following events are considered: "the object O_i is assigned to the location L_j". The frequency memory is a n-sized squared matrix (n is the size of the instance to be solved). This matrix is computed by adding all the solutions (encoded as matrices) along the TS walk. In this frequency matrix, the sum of any column or any row is equal to length of the TS walk (the number of iterations). Before starting a TS walk, the frequency memory is initialised to zero. Along the walk, it is updated by adding the visited solutions. At the end of the TS search, the frequency memory provides an information about the areas the TS has visited. Then, the TS sends the frequency memory and its best found solution to the adaptive memory.

3.2 The Adaptive Memory

The adaptive memory (AM) is the coordinator of the co-evolutionary meta-heuristic CO-SEARCH. At run time, the AM maintains a history of the search performed by the MTS. The AM is composed of three main parts: the global frequency memory, the set of the initial solutions of the TS and the set of the elite solutions (see Fig. 1).

The global frequency memory and the set of starting solutions are used by the GA for the diversification task. The global frequency memory is the sum of the local frequency memory yielded by the TS. For the intensification task, the set of elite solutions is used. This set gathers the best solutions the TS walks have found. It stores an information about the regions of best quality in the explored regions of the search space. The number of elite solutions is a parameter of the method. When a TS finishes, it communicates its best found solution to the adaptive memory. If the set of elite is not full, then the solution is added; else, and if the new solution is better than the worst elite, then the new solution is added and the worst is discarded. In any case, if the set of elite contains a solution its quality is equal to the quality of the new solution, no replacement is done. According to this last rule, the diversity of the elites is insured, and the adaptive memory represents a greater number of promising solutions.

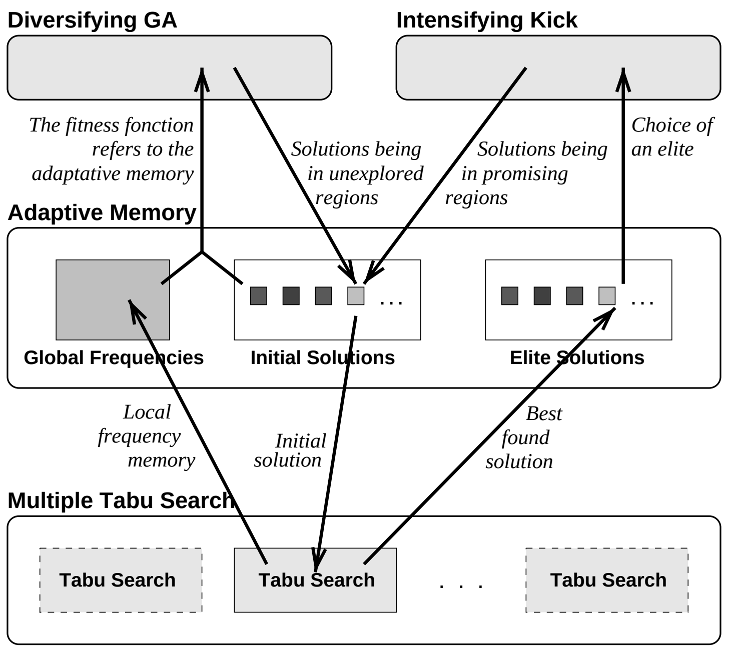

Fig. 1. The Genetic Algorithm cooperate with the Multiple Tabu Search via an Adaptive Memory which ensures the diversity of exchanged solutions.

3.3 The Diversifying GA

The diversifying agent, used in the CO-SEARCH, is a genetic algorithm. Genetic algorithms (GA) are meta-heuristics based on the natural evolution of living beings [6]. In the co-operative hybrid, the aim of the GA is to generate individuals in unexplored regions with regards to the AM. The GA is intrinsically efficient at the diversification task because the genetic operators generate scattered solutions. According to our experience [7], the GA is also able to adapt in a dynamic environment. It is helpful here because the AM, the GA refers to, is continuously being updated by the TS while the hybrid is running. When the GA has evolved a satisfying individual, that solution is given to a TS for being improved.

For the diversifying GA, there is a bi-objective function φ, composed of f and g to be minimised: f is relative to the frequency-based memory and g is relative to the set of initial solutions of the TS:

$$\varphi(u) = \alpha f(u) - g(u) \qquad f(u) = \sum_{e \in E} 1(e,u).F_e \qquad g(u) = \min_{v \in S} dist(u,v)$$

in which α is a coefficient which balances the influence of f and g, u is any individual in the population the GA is evolving, E is the set of the events considered to maintain the frequency memory F, F_e is the frequency stored in F relative to the event e, $1(e,u)$ is equal to 1 if the event e occurs in u else 0, S is the set of initial solutions of the TS.

The GA is running as follows: At each iteration, the genetic operators, crossover and mutation, are applied in competition on the best solution of the population. An other point of the population is randomly chosen, then used for the crossing operation. If a better solution than the best one is build, then it replaces the worst solution of the population. This GA, in which only one solution is replaced at each iteration, is in accordance with the steady-state model of GA proposed by Whitley [9].

For the QAP, we use the usual operators: the swap as mutation and the PMX as crossover [5]. The fitness function optimised by the diversifying GA is defined as follows:

$$\varphi(u) = \frac{1}{N} \sum_{i=1}^{n} F_{u(i)i} \quad - \quad \min_{v \in D} (\text{dist}(u,v))$$

where (F_{ij}) is the global frequency matrix, i and j denote respectively the objects and the locations, N is the total number of solutions "stored" in the global frequency memory. By dividing by N, the fitness is independent from the number of iterations of TS already done.

3.4 The intensifying kick

The diversifying agent yields starting solutions to the MTS. The intensifying agent we use in the CO-SEARCH is the Kick operator (KO). To generate a new solution, the KO refers to the set of elite solutions. The KO randomly chooses an elite and scrambles it, the objective being to do a small jump in the search space. If the KO scrambles too much the elite, it may build a solution out of the promising area in which the elite is. So, care should be taken not to scramble too much the elite to avoid the kick to act as a diversifying operator, or it leads to the opposite behaviour the KO is used for. For the QAP, we use a KO which consists in applying several times the swap operator.

4 Experiments

In order to show the influence of the diversifying agent and the intensifying agent, the metaheuristic CO-SEARCH is compared to a MTS. In this comparison, the MTS is considered as a CO-SEARCH in which the GA and the KO are discarded. However, in the MTS, the starting solutions are randomly generated and the

AM contains only the global best found solution. For this comparison, the same amount of CPU effort is used for both methods because the computational effort the GA and the KO are unsignificant with regard to the need of the MTS.

For the evaluation of each method, 1000 TS are done using quite small walks. In the MTS, the 1000 TS walks run simultaneously. In the CO-SEARCH, only a part of the TS walks are launched at the starting in order to provide partial results to update the AM. Actually, 100 TS are started at the beginning of the CO-SEARCH using uniformly distributed initial solutions. Then, when a TS achieves, the AM is updated and the GA or the KO provides an interesting solution for a new TS to restart with. The process is iterated until the 1000 TS are completed. So, the 900 last starting solutions are yielded by the GA or the KO alternately.

The TS walks are $100n$ iterations long, where n is the size of the instance to be solved. Before starting, a TS randomly sets the size of its tabu list between $\frac{n}{2}$ et $\frac{3n}{2}$. The population of the GA is randomly generated at the starting, its size is n. The number of generations used to build a diversified solution is n. The maximum size of the set of elite solutions is $\frac{n}{3}$, and the number of applications of the swap operation in the KO is $\frac{n}{3}$.

We have developed the co-evolutionary metaheuristic CO-SEARCH using the dynamic scheduling system MARS (Multi-user Adaptive Resource Scheduling) [8]. The MARS system harnesses idle cycles of time of a network of workstations. It supports the adaptive parallelism: It reconfigures dynamically the set of processors hosting the application with respect to the idle status of the workstations. A workstation is stated as idle when it is not loaded or its user is away (no one is using the console). The implementation of the parallel metaheuristic CO-SEARCH uses the master / workers paradigm. The master task, as usually done, do the management of the workers, however it also hosts the AM, the GA and the KO. The workers are sequential TS tasks. At run time, the master run on a single computer while the workers are dynamically distributed by the system MARS on the execution platform. So, some workers are running on the idle workstations while the other TS tasks are stored in the master until they are unfolded. When a node (workstation) becomes idle, the master unfolds a TS to it; When a node becomes busy, for it is loaded or owned, the TS task which is running on it is stopped (after the current iteration is completed) and retreated to the master. A checkpoint mechanism is available on the system MARS to prevent from failures or reboot during long runs. The platform we used to evaluate the co-evolutionary metaheuristic CO-SEARCH and the MTS is a network of heterogeneous computers: More than 150 workstations (PC-linux, Sun4-SunOS, Alpha-OSF, Sun-Solaris) and a cluster composed of 16 processors DEC-Alpha. A great deal of these workstations lies in the offices, the laboratories or the teaching rooms of the university.

All the testbed problems are extracted from the QAP library [2] which is a reference for people working on the QAP. The instances of QAP can be divided into several groups depending on their natures, from fully randomly generated instances to real world problems, from strongly structured instances to uniform instances. We propose a taxonomy using three types (type I, II and III). This taxonomy, based on the distribution of the local optima in the search space is the results of a study of the landscapes of the QAP we have carried out [1]. As the results of a heuristic search algorithm depend on this intrinsic nature [1], we have used QAP instances of different types to evaluate the parallel co-evolutionary metaheuristic CO-SEARCH.

			Best		**Average**	
Instance	**Type**	ψ **(QAPlib)**	MTS	CO-SEARCH	MTS	CO-SEARCH
Tai25a	I	1 167 256	**0.7361**	**0.7361**	**0.7361**	**0.7361**
lipa50a	I	62 093	**0.0000**	**0.0000**	**0.0000**	**0.0000**
Els19	II	17 212 548	**0.0000**	**0.0000**	**0.0000**	**0.0000**
Bur26d	II	3 821 255	**0.0000**	**0.0000**	**0.0000**	**0.0000**
Nug30	II	6 124	**0.0000**	**0.0000**	**0.0000**	**0.0000**
Tai35b	II	283 315 445	**0.0000**	**0.0000**	**0.0000**	**0.0000**
Ste36c	II	8 239 110	**0.0000**	**0.0000**	**0.0000**	**0.0000**
Sko64	III	48 498	**0.0000**	**0.0000**	0.0037	**0.0033**
Tai64c	-	1 855 928	**0.0000**	**0.0000**	**0.0000**	**0.0000**
Tai100a	I	21 125 314	0.6867	**0.3754**	0.7834	**0.5131**
Tai256c	I	44 759 294	0.2359	**0.1201**	0.2708	**0.1701**
Sko100a	III	152 002	0.0289	**0.0000**	0.0730	**0.0544**
Tai100b	III	1 185 996 137	0.2445	**0.0000**	0.3969	**0.1348**
Wil100	III	273 038	0.0205	**0.0029**	0.0352	**0.0085**
Tai150b	III	498 896 643	0.9246	**0.1903**	1.1276	**0.4393**
Tho150	III	8 133 484	0.0824	**0.0530**	0.1243	**0.0650**

Table 1. Results of the evaluations of the metaheuristics MTS and CO-SEARCH on instances of the QAP. The results are given using the format: $100 \times \frac{\psi\,(s)-\psi\,(QAPlib)}{\psi\,(QAPlib)}$ where s is the best found solution of the method and ψ is the cost function. The column **Best** shows the best result over 3 runs of CO-SEARCH or 10 runs of MTS. The column **Average** shows the average result over the runs. One can observe that the CO-SEARCH is more efficient then the MTS on almost all instances.

The evaluation compare the metaheuristics MTS and CO-SEARCH using quite the same amount of CPU time. To present the results, we distinguish between small instances ($n < 100$) and others (see Tab. 1). For small instances, both MTS and MTS provide the best known solution on almost all instances. Hence, the GA and the KO are no so useful for these instances, or the number of iterations should be reduced. On the opposite, for large instances, one can observe that the CO-SEARCH is very more efficient than the MTS. For 6 / 7 instances the CO-SEARCH provides a better solution; and on average, the CO-SEARCH is always better with a significant improvement for any type of instance. Moreover, the solutions found by the CO-SEARCH are high quality solutions in the sense that they are very close to the best known solutions. Hence, the CO-SEARCH is an efficient method, according to the CPU effort used. The CPU time used for the evaluation correspond to 1000 n iterations of TS, that is $1000\,\frac{n(n-1)}{2}$ times the evaluation of a solution. For example, for $n = 100$ (Tai100a, Wil100, etc, ...), the CO-SEARCH evaluates approximatively 500 millions solutions and the mean running time is 4 hours (depending on the load of the platform).

This evaluation shows the benefits of the co-evolution to solve large instances: The adding of the GA and the KO on the MTS lead to a better equilibrium between the intensification and the diversification which increase the efficiency of the metaheuristic.

5 Conclusion

In this paper, we have proposed CO-SEARCH, an efficient metaheuristic which combines the two advantages of parallelism: the computational power and the co-evolution. The computational power is used at the implementation level to reduce the running time; the co-evolution is used to well-balance the diversification and the intensification in the metaheuristic. The evaluations of the CO-SEARCH we have carried out and the comparison to the MTS (Multiple Tabu Search) shows that the co-evolution increases significantly the efficiency of the method. So, this study has reveals the interest of our approach based on the co-evolution paradigm to solve combinatorial problems.

References

1. V. Bachelet. *Métaheuristiques parallèles hybrides : application au problème d'affectation quadratique.* PhD thesis, Université des Sciences et Technologies de Lille, Villeneuve d'Ascq, France, December 1999.
2. R.E. Burkard, S. Karisch, and F. Rendl. Qaplib: A quadratic assignment problem library. *European Journal of Operational Research*, 55:115–119, 1991.
3. E. Çela. *The Quadratic Assignment Problem Theory and Algorithms.* Kluwer Academic Publishers, 1998.
4. F. Glover and M. Laguna. *Modern Heuristic Techniques For Combinatorial Problems*, chapter 3, pages 70–150. Blackwell Scientific Publications, 1992.
5. D. Goldberg and R. Lingle. Alleles, loci, and the traveling salesman problem. In *International Conference on Genetic Algorithms and their Applications*, 1985.
6. J.H. Holland. *Adaptation in natural and artificial systems.* The University of Michigan Press, Ann Arbor, MI, USA, 1975.
7. Bessiere P., Ahuactzin J.M., Talbi E-G., and Mazer E. The ariadne's clew algorithm: global planning with local methods. *IEEE International Conference on Intelligent Robots Systems IROS, Yokohama, Japan*, July 1993.
8. E-G. Talbi, J-M. Geib, Z. Hafidi, and D. Kebbal. Mars: An adaptive parallel programming environment. In R. Buyya, editor, *High Performance cluster computing*, volume 1, chapter 4. Prentice Hall PTR, 1999.
9. D. Whitley. GENITOR: A different genetic algorithm. In *Proc. of the Rocky Mountain Conference on Artificial Intelligence*, Denver, CO, USA, 1988.