Constraint-Based Local Search
for the Costas Array Problem

Daniel Diaz[1], Florian Richoux[2], Philippe Codognet[2],
Yves Caniou[3], and Salvador Abreu[4]

[1] University of Paris 1-Sorbonne, France
Daniel.Diaz@univ-paris1.fr
[2] JFLI, CNRS / UPMC / University of Tokyo, Japan
{richoux,codognet}@is.s.u-tokyo.ac.jp
[3] JFLI, CNRS / NII, Japan
Yves.Caniou@ens-lyon.fr
[4] Universidade de Évora and CENTRIA FCT/UNL, Portugal
spa@di.uevora.pt

Abstract. The Costas Array Problem is a highly combinatorial problem
linked to radar applications. We present in this paper its detailed mod-
eling and solving by Adaptive Search, a constraint-based local search
method. Experiments have been done on both sequential and parallel
hardware up to several hundreds of cores. Performance evaluation of
the sequential version shows results outperforming previous implemen-
tations, while the parallel version shows nearly linear speedups w.r.t. the
sequential one, for instance 120 for 128 cores and 230 for 256 cores.

1 Introduction

During the last decade, the family of Local Search methods and Metaheuristics
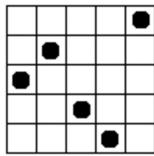has been quite successful in solving large real-life problems.

A generic Constraint-based Local Search method named Adaptive Search was
proposed in [4,5]. It is a metaheuristic that takes advantage of the structure
of the problem to guide the search and that can be applied to a large class
of constraints (*e.g.,* linear and non-linear arithmetic constraints and symbolic
constraints). Moreover, it intrinsically copes with over-constrained problems.

A parallel version with a multi-start approach requiring no communication
between processes has been defined in [7,3]. On classical CSP benchmarks from
CSPLib, this simple parallelization scheme gives good results, with a factor 50-70
speedup for 256 cores, but this is far from ideal speedup (*e.g.,* factor 256 speedup
for 256 processors), even for large problem instances. It is thus an open question
to know whether this is due to the classical (structured) CSP benchmarks used
or if this is a limitation of the method. In this paper we address the problem
of modeling a very combinatorial problem, with a low density of solutions the
Costas Array Problem (CAP) in the sequential version and we further investigate
if it scales up to a large number of processors and exhibits better speedups.
The CAP is an abstract problem that was motivated by a sonar application in

the 1960's but still has practical interest in radar and software-defined radio applications [2]. Experiments in solving the CAP by Adaptive Search on two parallel platforms, the Hitachi HA8000 supercomputer at University of Tokyo and GRID'5000, the French national Grid for the research, show nearly linear speedups w.r.t. the sequential version, for instance 120 for 128 cores and 230 for 256 cores.

The rest of this paper is organized as follows. Section 2 presents the Costas Array Problem. Section 3 presents the modeling of the Costas Array Problem within the Adaptive Search formalism. Section 4 details the experiments up to 256 cores on the HA8000 supercomputer and the GRID'5000 platform. Section 5 concludes the paper and briefly discusses about future work.

2   The Costas Array Problem

A Costas array is an $n \times n$ grid containing n marks such that there is exactly one mark per row and per column and the $n(n-1)/2$ vectors joining the marks are all different. We give here an example of Costas array of size 5. It is convenient to see the Costas Array Problem (CAP) as a permutation problem by considering an array of n variables $(V_1, \ldots, V_n)$ which forms a permutation of $\{1, 2, \ldots, n\}$. The Costas array above can thus be represented by the array $[3, 4, 2, 1, 5]$.

Historically, these arrays have been developed in the 1960's to compute a set of sonar and radar frequencies avoiding noise [6]. The problem to find a Costas array of size n is very complex since the required time grows exponentially with n. In the 1980's, several algorithms have been proposed to build a Costas array given n, such as the Welch construction and the Golomb construction [9], but these methods cannot built Costas arrays of size 32 and some higher non-prime sizes. Nowadays, after many decades of research, it remains unknown if there exist any Costas arrays of size 32 or 33. Another difficult problem is to enumerate all Costas arrays for a given size. Using the Golomb and Welch constructions, Drakakis *et. al* present in [8] all Costas arrays for $n = 29$. They show that among the 29! permutations, there are only 164 Costas arrays, and 23 unique Costas arrays up to rotation and reflection. There are constructive methods known to produce Costas arrays of order 24 to 29.

The Costas array problem has been proposed as a challenging combinatorial problem by Kadioglu and Sellmann in [10]. They propose a local search metaheuristic, *Dialectic Search*, for constraint satisfaction and optimization, and show its performance for several problems. Clearly this problem is too difficult for propagation-based solvers, even for medium size instances (*i.e.,* with n around $18-20$). Let us finally note that we do not pretend that using local search is better than constructive methods in order to solve the CAP. We rather consider the CAP as a very good benchmark for testing local search and constraint-based systems and to investigate how they scale up for large instances and parallel execution.

In [12], Rickard and Healy studied a stochastic search method for CAP and concluded that such methods are unlikely to succeed for $n > 26$. Although

their conclusion is true for their stochastic method, it cannot be extended to all stochastic searches: their method uses a restart policy which is too simple and they also used an approximation of the Hamming distance between configurations in order to guide the search which they recognized themselves not be be a very good indicator. However, they studied in this paper the distribution of solutions in the search space and shown that clusters of solutions tend to spread out from $n > 17$, which justify our multi-walk approach presented in Section 4 to reach linear speedup for high values of n.

3   Solving the CAP with Adaptive Search

The CAP can be modeled as a permutation problem by considering an array of n variables $(V_1, \ldots, V_n)$ which forms a permutation of $\{1, 2, \ldots, n\}$ (*i.e.*, implicit `alldifferent` constraint on variables V_i). A variable $V_i = j$ iff there is a mark at column i and row j. To take into account constraints on vectors between marks (which must be different) it is convenient to use the so-called *difference triangle*.

This triangle contains $n-1$ rows, each row corresponding to a distance d. The dth row of the triangle contains the differences $V_{i+d} - V_i$ for all $i = 1, \ldots, n-d$ (*i.e.,* the difference of values at a distance d). Ensuring all vectors are different comes down to ensure the triangle contains no

		3	4	2	1	5
$d=1$		1	-2	-1	4	
$d=2$			-1	-3	3	
$d=3$				-2	1	
$d=4$				2		

repeated values on any given row (*i.e.*, `alldifferent` constraint on each row). Here is the difference triangle for the Costas array given as example in Section 2.

In the Adaptive Search (AS) method, the way to define a constraint is done via error functions [4]. At each new configuration, the difference triangle is checked to compute the global cost and the cost of each variable V_i. Each row d of the triangle is checked one by one. Inside a row d, if a pair (V_i, V_{i+d}) presents a difference which has been already encountered in the row, the error is reported as follows: increment the global cost and the cost of both variables V_i and V_{i+d} by $ERR(d)$ (a strictly positive function). For a basic model we can use $ERR(d) = 1$ (to simply count the number of errors). Obviously a solution is found when the global cost equals 0. Otherwise AS selects the most erroneous[1] variable and will try to improve it.

Our AS sequential version has been tested over the CAP and compared to Dialectic Search (DS). AS outperforms DS on the CAP: for small instances AS is five times faster but the speedup seems to grow with the size of the problem, reaching a factor 8.3 for $n = 18$. [10] does not provide results for instances with $n > 18$.

CAP has also been used as a benchmark in the Constraint Programming community and we can compare with a CP Comet program made by Laurent Michel and based on the modeling in MiniZinc by Barry O'Sullivan[2]. As could

---

[1] i.e. the variable with the highest total error.
[2] `http://www.g12.cs.mu.oz.au/mzn/costas_array/CostasArray.mzn`

be expected, CP is much less efficient than local search, and this Comet program is about 400 times slower than AS for CAP19.

4   Parallel Implementation and Performance Analysis

We implemented a parallel version of AS using OpenMPI, an implementation of the MPI standard. Experiments and performance results on classical CSP benchmarks are described in [3]. The parallelization is straightforward and based on the idea of multi-starts and independent multiple-walks: fork a sequential AS method on every available cores. But on the opposite of the classical fork-join paradigm, parallel AS shall terminate as soon as a solution is found, not wait until all the processes have finished (since some searches initialized with "bad" initial configurations can take some time). Thus, some non-blocking tests are involved every c iterations to check if there is a message indicating that some other processes has found a solution; in which case it terminates the execution properly. This results in a high number of independent work units, a high CPU to I/O ratio, and no inter-process communication Three different testbeds were used on two platforms: The supercomputer HA8000 at the University of Tokyo (with a maximum of nearly 16000 cores) and the French national grid for research GRID'5000 (on two nodes at Sophia-Antipolis: Suno with 360 cores and Helios with 224 cores). Tables 1 & 2 show the execution times of the parallel executions on the HA8000 supercomputer and GRID'5000. Timings are given in seconds and are the average of 50 executions for each benchmark; they do not include the deployment time, negligible on big benchmarks.

Table 1. Speedups on HA8000, Suno and Helios for small instances of CAP

Platform	Problem	Time on 1 core	Speedup on k cores			
			32	64	128	256
HA8000	CAP 18	6.76	27.0	29.4	28.2	26.0
	CAP 19	54.54	29.6	54.5	75.7	99.2
	CAP 20	367.2	26.6	42.4	98.2	168
Suno	CAP 18	5.28	33.0	63.6	94.3	139
	CAP 19	49.5	36.1	83.9	121	226
	CAP 20	372	30.5	63.5	139	208
	CAP 21	3743	21.9	72.8	107	218
Helios	CAP 18	8.16	34.0	74.2	136	-
	CAP 19	52.0	22.6	59.8	130	-
	CAP 20	444	31.0	58.2	98.2	-

Behaviors on all three platforms are similar and exhibit very good speedups for larger instances. For $n = 21$ on Suno we have **a 218 times speedup on 256 cores** w.r.t. sequential execution. For the bigger instances CAP21 and CAP22, we present in Table 2 results for executions from 32 to 256 cores only, because the sequential time becomes prohibitive (*e.g.,* more than one hour on average

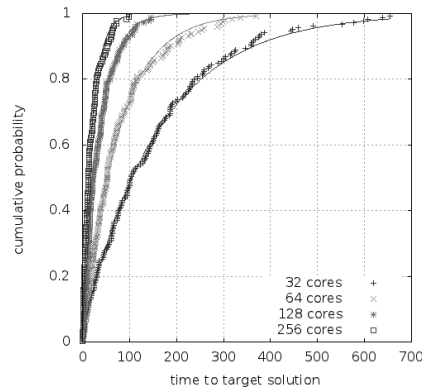**Table 2.** Speedups on HA8000, Suno and Helios for large instances of CAP

Platform	Problem	Time on 32 cores	Speedup on k cores		
			64	128	256
HA8000	CAP21	160.4	1.96	4.16	10.0
	CAP22	501.2	2.01	3.90	8.24
Suno	CAP21	171	3.32	4.90	9.94
	CAP22	731	1.92	3.66	7.09
Helios	CAP21	153	1.51	4.17	-
	CAP22	1218	2.34	5.53	-

for CAP21 and more than 10 hours for CAP22 on HA8000). We can see that on all platforms, **execution times are halved when the number of cores is doubled**, thus achieving ideal speedup. As a final result, we can now solve $n = 22$ in about one minute on average with 256 cores on HA8000.

Up to now, we focused on the *average* execution time in order to measure the performance of the method, but a more detailed analysis could be done. In [1,11], a method is introduced to represent and compare execution times of stochastic optimization methods by using so-called *time-to-target plots*. Observe that, for the CAP, the target value to achieve is obviously *zero*, meaning that a solution is found. It is then easy to check if runtime distributions could be approximated by a (shifted) exponential distribution of the form: $1 - e^{-(x-\mu)/\lambda}$. Then, according to [13], it is possible to achieve linear speedups by multiple independent walks if we have an exponential runtime distribution.

The following figure presents time-to-target plots for CAP 21 in order to compare runtime distributions over 32, 64, 128 and 256 cores.

Points represent execution times obtained over 200 runs and lines correspond to the best approximation by an exponential distribution. It can be seen that the actual runtime distributions are very close to exponential distributions. Time-to-target plots also give a clear visual comparison between instances of the same method running on a different number of cores. For instance it can be seen that we have around 50% chance to find a solution within 100 seconds using 32 cores, and around 75%, 95% and 100% chance respectively with 64, 128 and 256 cores.

5   Conclusion and Future Work

The CAP is a hard combinatorial problem for medium and large instances, too difficult to solve with classical propagation-based solver and we thus used

a constraint-based local search solver. We proposed a parallel version based on the idea of multi-starts and independent multiple-walks which naturally provides *Pleasantly Parallel* computations and appears viable as it exhibits a nearly linear speedup behavior. We are currently continuing our experiments by tackling larger instances and using more cores.

Future work will focus on more complex parallel execution methods with inter-processes communication, *i.e.,* in the dependent multiple-walk scheme, in order to further improve performance. The communication mechanism will be designed with the goals of (1) minimizing data transfers as much as possible, as we aim at massively parallel machines with no hierarchical memory, and (2) re-using some common computations and/or recording previous interesting crossroads in the resolution, from which a restart can be operated.

References

1. Aiex, R., Resende, M., Ribeiro, C.: Ttt plots: a perl program to create time-to-target plots. Optimization Letters 1, 355–366 (2007)
2. Beard, J., Russo, J., Erickson, K., Monteleone, M., Wright, M.: Combinatoric collaboration on costas arrays and radar applications. In: Proceedings of the IEEE Radar Conference, Philadelphia, USA, pp. 260–265 (2004)
3. Caniou, Y., Codognet, P., Diaz, D., Abreu, S.: Experiments in parallel constraint-based local search. In: EvoCOP 2011, 11th European Conference on Evolutionary Computation in Combinatorial Optimisation, Italy. Springer (2011)
4. Codognet, P., Díaz, D.: Yet Another Local Search Method for Constraint Solving. In: Steinhöfel, K. (ed.) SAGA 2001. LNCS, vol. 2264, pp. 73–90. Springer, Heidelberg (2001)
5. Codognet, P., Diaz, D.: An efficient library for solving CSP with local search. In: Ibaraki, T. (ed.) MIC 2003, 5th International Conference on Metaheuristics (2003)
6. Costas, J.: A study of detection waveforms having nearly ideal range-doppler ambiguity properties. Proceedings of the IEEE 72(8), 996–1009 (1984)
7. Diaz, D., Abreu, S., Codognet, P.: Parallel Constraint-Based Local Search on the Cell/BE Multicore Architecture. In: Essaaidi, M., Malgeri, M., Badica, C. (eds.) Intelligent Distributed Computing IV. SCI, vol. 315, pp. 265–274. Springer, Heidelberg (2010)
8. Drakakis, K., Iorio, F., Rickard, S., Walsh, J.: Results of the enumeration of costas arrays of order 29. Advances in Mathematics of Communications 5(3), 547–553 (2011)
9. Golomb, S., Taylor, H.: Constructions and properties of Costas arrays. Proceedings of the IEEE 72(9), 1143–1163 (1984)
10. Kadioglu, S., Sellmann, M.: Dialectic Search. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 486–500. Springer, Heidelberg (2009)
11. Ribeiro, C., Rosseti, I., Vallejos, R.: Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms. Journal of Global Optimization 17, 1–25 (2011) (published online August 17, 2011)
12. Rickard, S., Healy, J.: Stochastic search for costas arrays. In: Proceedings of the 40th Annual Conference on Information Sciences and Systems, Princeton, NJ, USA (March 2006)
13. Verhoeven, M., Aarts, E.: Parallel local search. Journal of Heuristics 1(1), 43–65 (1995)