POSL: A Parallel Oriented Solver Language

(detailed plan)

Alejandro REYES-AMARO
Éric MONFROY, Florian RICHOUX

April 11, 2016

Contents

Chapter 1

Introduction

The *Introduction* of the work is presented. We describe the target problem (the formal definition will be in the next chapter), and the approaches implemented so far to solve them. The necessity of a new approach to exploit the new era of parallelism is introduced.

In this section are presented the goals of the thesis, and POSL is introduced as a new parallel approach including others and novel features. Finally, we describe the structure of the document.

Chapter 2

Overview of Combinatorial Optimization Problems and methods to solve them

This chapter presents an overview to the state of the art of *Combinatorial Optimization Problems* and different approaches to tackle them.

2.1 Combinatorial Optimization

We introduce the definition of a *Constraint Satisfaction Problem* (*CSP*), where we concentrate our main efforts. We give some examples: *Resource Allocations* [1], *Task Scheduling* [6], *Master-keying* [5], *Traveling Salesman*, *Knapsack Problem*, among others.

2.2 Constraint propagation

Constraint propagation techniques are deterministic methods to attack these kind of problems, but in some cases they are incapable to solve them [4].

2.3 Meta-heuristic and Hyper-heuristic methods

However, we can not solve some *CSPs* only applying constraint propagation techniques. It is necessary to combine them with other methods. In this chapter is presented an overview in the field of *meta-heuristic* methods [3], nature-inspired algorithms divided in two groups:

1. *Single Solution Based:* more exploitation oriented, intensifying the search in some specific areas. (We will focus our attention on this first group)

2. *Population Based:* more exploration oriented, identifying areas of the search space where there are (or where there could be) the best solutions.

2.3.1 Single Solution Based Meta-heuristic

Methods of the first group are also called *trajectory methods*, and they are based on choosing a solution taking into account some criterion (usually random), and they move from a solution to his *neighbor*, following a trajectory into the search space.

2.3.2 Population Based Meta-heuristic

Also there exist heuristic methods based on populations. These methods do not work with a single solution, but with a set of solutions named *population*.

2.3.3 Hyper-heuristic Methods

Hyper-heuristics are automated methodologies for selecting or generating heuristics to solve hard computational problems.

2.4 Hybridization

The *Hybridization* approach is the one who combine different approaches into the same solution strategy, and recently, it leads to very good results in the constraint satisfaction field, some of them presented in this section.

2.5 Parallel computing

The evolution of computer architecture is leading us toward massively multi-core computers for tomorrow, composed of thousands of computing units. A parallel model to solve *CSPs* is the core of this work, and its advances are presented in this section.

2.6 Solvers cooperation

The interaction between solvers exchanging some information is called *solver cooperation* and it is very popular in this field due to their good results, presented in this section.

2.7 Parameter setting techniques

Most of these methods to tackle combinatorial problems, involve a number of parameters that govern their behavior, and they need to be well adjusted, and most of the times they depend on the nature of the specific problem, so they require a previous analysis to study their behavior [2]. That is way another branch of the investigation arises: *parameter tuning*. It is also known as a meta optimization problem, because the main goal is to find the best solution (parameter configuration) for a program, which will try to find the best solution for some problem as well. Finally, this chapter presents an overview of the progresses in the field of *parameter settings*.

The are tow classes to classify these methods:

1. *Off-line tunning*: Also known just as parameter tuning, were parameters are computed before the run.

2. *On-line tunning*: Also known as parameter control, were parameters are adjusted during the run, and

2.7.1 Off-line tunning

The technique of parameter tuning or off-line tunning, is used to computed the best parameter configuration for an algorithm before the run (solving a given instance of a problem), to obtain the best performance.

2.7.2 On-line tunning

Although parameter tunning shows to be an effective way to adjust parameters to sensibles algorithms, in some problems the optimal parameter settings may be different for various phases of the search process. This is the main motivation to use on-line tuning techniques to find the best parameter setting, also called *Parameter Control Techniques*.

Chapter 3

First attempts

Aiming for the right direction in order to find the proper approach, we have done some previous work that we present in this chapter.

3.1 Relaxation model for discrete *CSPs*

On a first attempt to tackle the problem of reducing the search space of a *CSP*, we model it through a continuous optimization problem, and then, applying efficient methods to solve it. This way we do not reach an optimal solution, but an approximation of it. The new variables domain would be the neighborhood of the found approximation.

3.2 Domain Split

A way to tackle huge combinatorial problems in parallel is to split the search space. In this section, the *problem subdivision* approach was adopted to divide the domain of a given problem, in this particular case, to solve the *k-medoids problem* in parallel. [1]

3.3 Tunning methods for local search algorithms

Another performed study was applying the PARAMILS tool in order to find the optimum parameter configuration to *Adaptive Search* solver. PARAMSILS (version 2.3) is a tool for parameter optimization for parametrized algorithms. It is an open source program (project) in *Ruby*, and the public source include some examples and a detailed and complete User Guide with a compact explanation about how to use it with a specific solver.

[1]This work falls within the framework of the *Ulysses* project between France and Ireland

Chapter 4

A Parallel-Oriented Language for Modeling Constraint-Based Solvers

In this chapter we introduce POSL as our main contribution and a new way to solve *CSPs*. We resume the its characteristics and advantages, and we get into details in the next sections. We describe a general outline to follow in order to build parallel solvers using POSL, and following each step is described in details.

4.1 First Stage: Modeling the target benchmark

In this stage we explain formally our modeling process of a benchmark to be solved (or study) through POSL. We explain how to make use of the already existing models or to create new benchmarks using the basic layer of the framework (in C++) making a proper usage of the object-oriented design.

4.2 Second Stage: Creating POSL's modules

There exist two types of basic modules in POSL: *operation modules* and *open channels*. An *operation module* is a function receiving an input, then executes an internal algorithm, and returns an output. An *open channel* is also a function receiving and returning information, but in contrast, the *open channel*can receive information from two different ways: through parameter or from outside, i.e. by communicating with a module from another solver.

4.2.1 Operation Module

In this sub-section we expose the definition and the characteristics and the details of the *operation module*, and give some examples. We explain how to create new *operation modules* using the basic layer of the framework (in C++) making a proper usage of the object-oriented design.

4.2.2 Open channel

In this sub-section we expose the definition and the characteristics and the details of the *open channel*, and give some examples. We explain how to create new *open channels* using the basic layer of the framework (in C++) making a proper usage of the object-oriented design.

4.3 Third Stage: Assembling POSL's modules

In this stage a generic strategy is coded through a operator based language: POSL. We call this the *computation strategy*. The operators are parametrized functions receiving information and allow interactions between modules (*operation modules* and *open channels*). In this section are defined the operators and some examples are presented. We explain how to create new *operators* using the basic layer of the framework (in C++) making a proper usage of the object-oriented design.

4.4 Forth Stage: Creating POSL solvers

With operation modules and open channels already assembled through the *computation strategy*, we can create solvers by instantiating modules. POSL provides an environment to this end and we present the procedure to use it.

4.5 Fifth Stage: Connecting the solvers

Once we have defined our solver strategy, the last step is to declare the *communication channels*, i.e. connecting the solvers each others. In this last stage, POSL provides to the user a platform to easily define cooperative meta–strategies that solvers must follow (POSL meta–solvers). The steps to create *communication channels* through *communication operators* are explained and some examples are presented. We explain how to create new *communication operators* using the basic layer of the framework (in C++) making a proper usage of the object-oriented design.

4.6 An step-by-step POSL code example

In this section we summarize all the steps to build a POSL meta–solver through an real example, providing schemes to make more comprehensive the process.

Chapter 5

Study and evaluation process

In this chapter we expose all details about the process of evaluation of POSL, i.e. all the experiments we perform. For each benchmark, we explain also the strategy (or strategies) used in the solving (evaluation) process.

5.1 Target benchmarks

We present all the benchmarks that we used to evaluate POSL. We give the formal definition of them and their main characteristics, explaining also our interest in them. The studied strategies used to evaluate POSL through this benchmark are presented and explained in details.

5.1.1 Social Golfers Problem

We present in this sub-section the definition, characteristics and some implementation details of the *Social Golfers Problem*. Studied strategies used to evaluate POSL through this benchmark are presented and explained in details.

5.1.2 Costas Array Problem

We present in this sub-section the definition, characteristics and some implementation details of the *Costas Array Problem*. Studied strategies used to evaluate POSL through this benchmark are presented and explained in details.

5.1.3 N-Queen Problem

We present in this sub-section the definition, characteristics and some implementation details of the *N-Queen Problem*. Studied strategies used to evaluate POSL through this benchmark are presented and explained in details.

5.1.4 Golomb Ruler Problem

We present in this sub-section the definition, characteristics and some implementation details of the *Golomb Ruler Problem*. Studied strategies used to evaluate POSL through this benchmark are presented and explained in details.

5.1.5 All Interval Problem

We present in this sub-section the definition, characteristics and some implementation details of the *All Interval Problem*. Studied strategies used to evaluate POSL through this benchmark are presented and explained in details.

5.1.6 ...

Maybe others...

Chapter 6

Analysis of results

In this chapter we explain the used environments were we run the experiments (description of my desktop machine, *Curiosiphi* server, and eventually *Grid5k*). We describe all the experiments and we expose a complete analysis of the obtained result.

Chapter 7

Conclusions and future works

We resume our work, emphasizing on our contribution and obtained results, and we expose the conclusions of the work. We also discus future branches to follow that can be derived from our work. Finally we give our conclusions.

Appendix A

POSL **strategies**

In this appendix we present some secondaries *computation strategies* (written in symbolic POSL code) used during our work, not directly related with our main results.

Appendix B

POSL **code**

In this appendix we present the POSL code used during our work in all the main performed experiments.

Appendix C

Complete results

In most of the cases, the complete result tables show interesting behaviors of POSL, like for example, the percentage of times when the communication was effective (a solver was able tu find a solution thanks to the received information). For that reason we present in this Appendix this results.

Appendix D

POSL **Documentation**

A complete and detailed documentation of the code is presented in this Appendix.

Bibliography

[1] Mahuna Akplogan, Jérome Dury, Simon de Givry, Gauthier Quesnel, Alexandre Joannon, Arnauld Reynaud, Jacques Eric Bergez, and Frédérick Garcia. A Weighted CSP approach for solving spatio-temporal planning problem in farming systems. In *11th Workshop on Preferences and Soft Constraints Soft 2011.*, Perugia, Italy, 2011.

[2] Mauro Birattari, Mark Zlochin, and Marrco Dorigo. Towards a Theory of Practice in Metaheuristics Design. A machine learning perspective. *RAIRO-Theoretical Informatics and Applications*, 40(2):353–369, 2006.

[3] Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82–117, jul 2013.

[4] Christian Bessiere. Constraint Propagation. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, chapter 3, pages 29–84. Elsevier, 1st edition, 2006.

[5] Wolfgang Espelage and Egon Wanke. The combinatorial complexity of masterkeying. *Mathematical Methods of Operations Research*, 52(2):325–348, 2000.

[6] Louise K. Sibbesen. *Mathematical models and heuristic solutions for container positioning problems in port terminals*. Doctor of philosophy, Technical University of Danemark, 2008.