On Defining Design Patterns to Generalize and Leverage Automated Constraint Solving

Thiago Serra

PETROBRAS – Petróleo Brasileiro S.A.
Avenida Paulista, 901, 01311-100, São Paulo – SP, Brazil
thiago.serra@petrobras.com.br

Abstract. This position paper reflects on the generalization of adaptive methods for Constraint Programming (CP) solving mechanisms, and suggests the use of application-oriented descriptions as a means to broaden CP adoption in the industry. We regard as an adaptive method any procedure that modifies the behavior of the solving process according to previous experience gathered from similar cases. Despite its design being much of a creativity matter, many patterns emerge from the comparison of existing methods. As a starting point, we propose a framework with design patterns for learning time, search modification, and case discrimination. Those patterns provide a glimpse of the circumstances in which certain design choices are better suited than others, and thus define a language to handle and address application domains with diverse needs.

Keywords: Algorithm Selection, Hardness Models, Autonomous Search

1 Introduction

The reliability of a solver is related to its robustness to deal with varied types of problems. Constraint Programming (CP) is regarded as a valuable technique due to its expressive modeling capabilities. However, it is crucial that its solving performance matches properly the expectations of such expressiveness. On the one hand, it has been observed that problems with different characteristics are better suited for different CP search algorithms [13]. On the other hand, search automation has been seen by some authors as an important step towards ease of use, and thus broadening CP's audience [49, 23]. Considered together, those factors not only foster the development of black-box solvers – in which search definition is abstracted from the modeler [23] – but also praise for a design that makes a better use of benchmark sets of problems and solver feedback. Namely, rely on them to learn how to select the best algorithm in a real situation instead of electing an anticipated overall winner. As observed by Selman et al. [53], a misleading use of benchmarks may produce wrong conclusions about the hardness of a problem and, conversely, about the effectiveness of a given algorithm. Such idea is also backed by the competitive edge attributed to adaptivity in recent solver competitions [65, 46]. Hence, there is ample evidence that the CP solving process can be undertaken in a more robust and accessible way if adaptive methods are carefully designed to exploit the available data, and used in the attempt to select the most suitable search procedure for each case.

Adaptivity has been approached in the literature from a variety of perspectives. Research groups enrolled in the project of different solvers have each developed slightly different mechanisms. Some of them even recognize the orthogonality of their efforts for dealing with diverse aspects of the solving process, e.g., [45]. Simultaneously, others engaged with understanding the performance degeneracy phenomena developed a comprehensive insight to put those mechanisms some steps further, e.g., [53, 39, 21]. In all cases, their ultimate goal is to figure how to leverage the subtleties of each case for the sake of a better performance.

Attempting to generalize and categorize such developments, a formal taxonomy to classify adaptive search mechanisms according to their design was first proposed by Hamadi et al. [25]. It defines a hierarchical scheme consisting of three layers: (i) when search modifications occur; (ii) which information induces them; and (iii) how search is being modified. We have introduced an alternative characterization in [54] to decompose and extend such scheme into orthogonal design decisions. For each of them, a number of patterns were described to fit the needs of varied situations, thus providing a rationale for the existence of opposing trends on the topic. One of the intended contributions of this work consists of extending such characterization by prescribing when each of those patterns is more appropriate, and situating them in the literature. Far from being definitive, this extension aims to demonstrate the suitability of an application-oriented description to characterize and differentiate adaptivity as currently conceived.

From a broader perspective, this work aims at evidencing how the use of such a characterization would be beneficial to leverage performance by requiring that modelers explicitly define conditions and expectations for each application. At first, it might appear contradictory to praise for the definition of strategies for adaptivity if those strategies are intended to abstract the outline of search, and thus reduce the burden on modelers. However, the conditions of use are better known by the application stakeholders than by the solver designers, as opposed to the implementation and parametrization of search algorithms. Another possible criticism to this proposition is that such a depiction is prone to be limiting, what would hamper creativity. Minding that such flaw would occur if the depiction became overly detailed, we managed to keep our characterization at the highest possible level. Hence, we hope to raise debate on how to turn general insights of adaptive methods into best practices to design strategies for CP solvers.

Our argument is further detailed in the next two sections, which can be skipped in a first reading. Section 2 justifies the importance and describes the context for the use of adaptive methods in constraint solving. Section 3 revises standard references on the topic while portraying their commonalities and differences according to a pattern-based model. In the sequence, Section 4 presents final remarks on developing and benefiting from an application-oriented characterization. They are based on the analysis of similar efforts, such as Hooker's work in the development of an integrated perspective for optimization methods [29], and the consolidation of design patters in the field of software engineering.

2 Constraint Programming and Adaptive Methods

The solving process in CP is concerned with finding a solution to a Constraint Satisfaction Problem (CSP). Each CSP can be described by a tuple (V, D, C). A solution to a CSP must be such that each variable $V_i \in V$ has a value from its domain $D_i \in D$, and those assignments satisfy all constraints in the set C.

The search for a solution is said to be complete if it consists of a systematic procedure capable of exhausting the search space in finite time. In a complete search, the solving process is usually based on backtracking algorithms [14]. This type of algorithm tries to find a feasible solution by orderly assigning a value to each variable, and reversing such assignments if a constraint is violated [6]. Such algorithms can be further decomposed into a framework of varied mechanisms for propagation, retraction and branching heuristics [17]. *Propagation* consists of using the constraints among variables to infer how to prune their domains, and thus reduce the search effort. It relies on filtering algorithms, each of which designed to achieve a given level of consistency [4]. *Retraction* operates to roll-back from dead-ends where no solution can be found. Algorithms for retraction vary between being cheap but conservative, and requiring more effort but reaching earlier a point where the cause of the conflict can be dismissed [6]. *Branching heuristics* are assignment prioritization criteria based on simple but diverse rules intended to anticipate search conclusion [17]. They are usually designed to select first variables whose assignment is more likely to reduce the domain of the remaining variables, and values that are more likely to reach a full assignment without conflicts [50]. As a fourth component, *restart strategies* have been used as policies to reset the process by interrupting the search and starting it over with a different configuration. They aim at preventing bad decisions at early steps from ruining search performance [24], and at improving the probability of success of randomized algorithms [1]. In an incomplete search, it is possible to use variations of the former elements as well as stochastic and local search methods, which have an experimental record of success in cases where exhaustive search is not viable [30]. They can also be decomposed into framework schema, e.g., [38, 43].

Early developments in CP search focused on a generic quest for robustness [42] that has been gradually replaced by adaptive procedures. Still, they were valuable to establish an intuitive tradeoff between inference and retraction mechanisms, outline successful heuristics, and explore promising extensions [6]. However, the generality of those findings also means that they are not able to indicate the most suitable configuration for specific applications and purposes.

Such need for a case-based configuration of solving mechanisms is addressed by adaptive methods. Their cornerstone is attributed to Rice [55], who formally defined the Algorithm Selection Problem (ASP) [51]. It comprises the mappings of a problem space into an algorithm space, and of the latter into a performance measure space. Such idea was later legitimated by the No Free Lunch (NFL) theorems. They state that no single algorithm performs best on all range of problems of search [61] or optimization [62]. A similar concept is presented by Corne and Reynolds [12] as the algorithm footprint, which represents the instance space where each algorithm performs comparatively better. Furthermore, Leyton-Brown et al. [39] praise for a design of algorithms based on the characterization of instances which are not well solved by the algorithms already known. While revisiting the subject, Smith-Miles [55] observed that the awareness about Rice's work was restricted to the field of meta-learning in the machine learning community for a long while. However, many related projects have been developed in diverse areas regardless of referring to it [55]. A discussion of selected topics related to implementing and abstracting the metaphoric description given to some of those methods was recently reported by Battiti et al. [5]. Hence, the topic has a sound theoretical basis to which several developments are being performed.

In the case of CP, at least three research directions have contributed to the study of algorithmic performance, and thus to support the use of adaptive methods. First, there is the development of search hardness models that associate machinery cost to the use of each algorithm on each problem, e.g., [32, 40]. They are based on structural properties of CSPs such as density of constraints among pairs of variables and restrictiveness of tuples that satisfy each constraint [21]. Second, typical-case complexity models have been used to break down the problem space of the CSP, which is NP-complete in the general case [14]. Those models suggest that the space can be divided into easy-hard-easy phase transitions, proven to be an algorithm-independent problem invariant [53]: it is easier to find a solution for an underconstrained problem or to prove that none exists for an overconstrained one, but it is harder to explore the constrainedness region between both [21]. Third, some researchers have been exploring conditions under which a feasible problem becomes cheap or costly to solve, thus providing guidance to pursue or avoid specific conditions. On the one hand, some classical CSPs possess small variable sets that if assigned leave the remaining problem easy to solve. They are known as backdoor sets [60]. On the other hand, the cost of some algorithms within underconstrained regions becomes irregular and exceptionally higher due to the exploration of large inconsistent trees [21]. This phenomenon is known as heavy-tailed behavior. It is inherent to some combinations of heuristics and problems [60]. Altogether, those findings explain why the worst-case is often unrealistic for CP, but also endorse the use of robust strategies to avoid it.

3 Adaptive Search Patterns

The aim of the current description is to detail the characterization from [54] in regard to the application and reported use of each observed pattern. More generally, we strive to understand adaptive search methods as a theoretical triplet of *state*, *choice* and *result*. It differs from Rice's ASP [51] in that state and choice can be represented on varied abstraction levels, for which problem and algorithm are one of the available options. Three types of patterns are considered to depict the way in which those elements can be used to design adaptive search methods. The first type of pattern refers to the timing of the learning process, which may involve the whole triplet. The second type describes the kinds of modifications than can be made, thus comprising the element of choice alone. The third type depicts the discrimination of cases, which stands for a combined use of state and result to associate each choice with an expected outcome. The focus on those pattern types is based on the premise that their use is mandatory if one intends to design a truly adaptive mechanism. In spite of that, some of the approaches used to illustrate the application of each pattern do not comply with such rule. That happens either because they consist of early work on the topic, or because they have focused on specific issues instead. Therefore, the depiction here present is not aimed to develop a comprehensive synthesis of what has been done, but to use previous work to evidence how adaptivity is currently conceived.

Such pattern-based description is inspired on the design patterns for object-oriented software development. Gamma et al. [19] described pattern types to abstract the process of creating new objects, to compose objects and extend their functionalities, and to distribute behavior between objects. Each of them represents a solution guideline to a recurring design issue associated with certain circumstances. Likewise, the need for an adaptive approach might arise in situations with varied availability of data, processing time, and knowledge about the problem domain. Our depiction of pattern types to represent the use of adaptive methods somewhat resembles a behavioral pattern known as template method. However, it does not fully comply with such design pattern because the patterns described for each type cannot be interchangeably used: their inputs and outputs with respect of the rest of the system are different. In resemblance to the organization observed in [19], each adaptive search pattern will be presented in terms of its intent, cases of application, and incidence in the literature.

3.1 Learning Time Patterns

The adaptation of search relies on some learning process whose timing may vary. Its outcome will provide information to endorse the choices for modifying the search behavior. That information consists of a mapping of static and dynamic information about each case onto performance metrics of interest. If such process is totally absent, the search will either rely on a standard algorithm or be regarded as a problem-specific required input. In the past, such directives used to be requested more often and extensively, e.g., the former search syntax of the Optimization Programming Language (OPL) [27]. However, recent developments related to CP modeling languages and their solvers, such as OPL [50, 38] and Comet [26, 45], have been pushing the practice towards an autonomous search. The patterns below refer to the learning time with respect to the search.

Offline

Intent: Search is tuned once and prior to use. The adaptation relies on an initial set of instances, which are expected to be a representative example of real use.

Application: The problems being solved do not vary much with respect to the set of examples. Response time is important during the use, but plenty of time is available for training before deploying the solver.

References: The use of a preliminary training setup dates back to Minton's [44] work, one of the first adaptive approaches for CP. It consists of the Multi-Tactic Analytic Compiler (Multi-TAC), which selects a search algorithm for each problem after evaluating some algorithms in a sample set of instances. In the long-term project of the Adaptive Constraint Engine (ACE) [17, 16, 48], a separate training for selecting the most appropriate heuristics for branching is made for three phases of the search. Those phases differ by the amount of assignments already made. The training occurs within the solving process of the first instances, but it is restarted if the performance gets worse than expected for a while. Training procedures are also used in the development of empirical hardness models for complete search in [65, 40], and stochastic search in [33]. In addition, schedules of solvers are built with training in the works of Yun and Epstein [66], and Kadioglu et al. [34]. Provided that there is a comprehensive set of instances and their associated performance on a broad range of algorithms, it was observed in [34] that there is only a minor gain with the acquisition of further data from routine use, and thus the incurred performance overhead can be avoided.

Online

Intent: Search is tuned from the beginning at each run. The adaptation is made according to information gathered during the search.

Application: The amount of data for preliminary tests is insufficient to draw any conclusion. An overhead in the response time to guarantee performance reliability is acceptable, or there is no time for training out of the solving process.

References: There is a resemblance among some approaches using online learning and those of the offline learning. That is the case of [11, 65, 46], since all of them use a setup time to test among candidate algorithms at the start of each search. However, in most of the cases there is a continuous storage of the impact of previous decisions in the search performance. Such impact is recorded as a combination of the search state and the choices previously made. It is then used to select branching heuristics in [9, 50, 3], and restart strategies in [37, 18].

Interleaved

Intent: Search is progressively tuned in the interval between consecutive runs. The adaptation relies on experiments with data from previous instances.

Application: The characteristics of the data from routine use are unknown, but they are expected to vary less than any set provided beforehand. Response time is critical, or there is plenty of processing time available when the solver is idle.

References: Despite being promising for applications running on dedicated machinery, this pattern has been comparatively less explored. It was strongly praised by Arbelaez and Hamadi [2], which coined it as continuous search. Using the solver idle time for extensive experiments with previous instances, they observed an incremental improvement in performance. The use of information from previous runs is also described in some approaches of Case-Based Reasoning (CBR) [41, 20]. In those works, each instance and the algorithm that solves it are stored as a case afterwards, and similar cases are retrieved to solve a new instance of the problem. It is worth observing that the concept beneath many offline learning approaches could be easily adapted to this pattern.

3.2 Search Modification Patterns

The adaptation of the solving process depends on the degrees of freedom purposely left in the search design. It is by means of them that it is possible to experiment with different choices during the learning process, and thus to modify the search behavior afterwards. To a certain extent, the change in the search behavior depends on the level of abstraction at which the solving process can be modified. They can range from the evaluation of algorithms down to the inner assembly of one of them. On either case, a single procedure or a schedule of procedures can be used in the attempt to solve the problem. The patterns that follow depict the composition of portfolios of algorithms, the selection of a single algorithm, and the use of portfolios or selectors on components of a framework.

Algorithm Portfolio

Intent: A number of procedures concur for the solving runtime. Each of them is expected to explore the search space in a different way.

Application: The performance of neither algorithm dominates the performance of the others for the type of problem being solved. The performance variability is large enough to compensate for the overhead of switching among algorithms.

References: The portfolio composition was first described by Huberman et al. [31] as a means to establish a tradeoff between benefit and risk, respectively defined as the expected search cost and variance. Using non-deterministic algorithms, Gomes and Selman [22] built those portfolios with a number of copies of each algorithm being executed in parallel or interleaved. In what followed, the concept was generalized for varied objectives. Leyton-Brown et al. [40] made a portfolio of algorithms for which the easier instances are distinct. In the Hydra solver [64], such idea was extended to characterize features of instances that are not well solved by the portfolio, and then include parameterized algorithms to address them. The CPHYDRA solver [46] interleaves different search algorithms to maximize the expected number of problems solved within a given amount of time. Streeter et al. [58] aimed to build a schedule of solvers that minimizes the overall runtime to solve a given set of problems, which was also used by Yun and Epstein [66]. Carchrae and Beck [11] described a portfolio with interleaved execution to solve optimization problems such that the time granted for each algorithm depends on how much it has contributed to find better solutions so far.

Algorithm Selector

Intent: A single procedure is selected to perform the search. It can be changed if the observed performance becomes worse than expected.

Application: An algorithm has a performance dominance over the others in most of the expected cases, or the performance variability is irrelevant if compared to the cost of using many algorithms at a time and switching among them.

References: This pattern is also described in the literature as a "winner take all" approach [65]. It is used by some CBR approaches [41, 20], and in the Multi-TAC solver [44]. Such idea is also explored by Hentenryck and Michel [26], and led Monette et al. [45] to develop AEON: a classification algorithm that selects the most specialized search algorithm known to solve the problem according to its structure. Hardness models are used to select one search algorithm among many in the work of Hutter et al. [33]. A slightly diverse selection is made using the approach described by Borret et al. [8][1] as the Quickest First Principle (QFP): the switch is made on a chain of progressively more robust but costly algorithms.

Algorithmic Framework

Intent: A search procedure is composed by choosing components upon a unified framework. Each type of component has a distinct role in the search process.

Application: The available algorithms roughly comply with a standard framework, thus allowing the use of a single parameterized algorithm. The benefit of manipulating such parameters compensates for the incurred performance losses.

References: As pointed out in Section 2, search in CP is often based on a standard backtracking framework. Most of the approaches described in the literature have been only adapting the components of such framework, either with a selector or a portfolio builder. This adaptive pattern actually describes an instance of the template method design pattern; and each role with interchangeable options of components is an instance of the strategy design pattern [19].

The selection of branching heuristics is one of the most prolific topics regarding adaptive methods for CP. Many authors have approached the selection among standard heuristics. In the Multi-TAC solver [44], they are either prioritized or weighted. In the ACE [17], those which are always right in a certain

[1] This work was recently republished with a few changes in [7].

domain are prioritized, being the remainder weighted according to their effectiveness. A Support Vector Machine (SVM) is used by Arbelaez et al. [3] to map the effect of using each heuristic on each search state.

Some other approaches have been developing new types of heuristics aimed at addressing performance variability issues more closely:

- *Combination of features.* The combination of features used by standard heuristics was tested to develop new heuristics in the ACE solver [17].
- *Impact-based heuristics.* Developed by Refalo [50], impact-based heuristics are aimed at ordering the variables to maximize search space reduction due to immediate propagation, and at provoking the opposite effect in the value ordering. Those heuristics are directed by the recorded measures of impact due to previous branching decisions. The incorporation of other measures was discussed by Cambazard and Jussien [10]. The use of the variance in combination with the average impact was approached by Kadioglu et al. [36].
- *Weighting of constraints.* Boussemart et al. [9] proposed to associate a weight to each constraint, which is proportional to the number of dead-ends that it causes in the search. Variables that are present in heavier constrains are then selected earlier for assignment during the search.
- *Solution counting.* By counting the number of solutions of each constraint, Zanarini and Pesant [67] targeted to prioritize assignments that are more frequent in the solutions of the most restrictive constraints.

Finally, there were authors that strived to understand how to make a better choice when selecting branching heuristics. Wallace [59] observed two important factors for changing search performance due to branching criteria: selecting heuristics (i) to anticipate immediate failure; and (ii) to propagate towards future failure. Furthermore, Hulubei and O'Sullivan [32] found that some ordering heuristics combinations are inherently heavy-tailed while others avoid thrashing.

Restart strategies were explored for varied degrees of availability of information about algorithm performance. Huberman et al. [31] developed optimal strategies for restarting with full or no information about algorithm performance. Kautz et al. [37] as well as Gagliolo and Schmidhuber [18] explored strategies for adapting according to partial information being acquired during the search. It is worth of note that a restart strategy is also considered as a special case of portfolio in which the same algorithm is used many times.

Adaptive mechanisms for propagation have also been explored to some extent. Early work was reported by El Sakkout et al. [15] on varying the filtering algorithms during the search to achieve a same consistency level with less effort. More recently, some approaches have been trying to define the level of propagation of each constraint. Stergiou [57] evaluated using simple dynamic heuristics, while Stamatos and Stergiou [56] evaluated using preliminary search tests for that matter. Wu and Beek [63] described a portfolio of backtracking algorithms that actually corresponds to an algorithmic framework, since the difference between those algorithms was the level of propagation used by each of them. In addition, Schulte and Stuckey [52] explored monitoring potential perturbations to the consistency level of each constraint in order to decide when and how to apply its propagators on a chain-based fashion. O'Sullivan [47] postulated as future work the automatic synthesis of filtering algorithms for global constraints.

Focusing on a local search framework instead, Laborie and Godard [38] extended the principles of the algorithm portfolio developed by Carchrae and Beck [11] for scheduling optimization. Using the Large-Neighborhood Search (LNS) metaheuristic, their approach consisted of modifying the probability of selection of neighborhood operators that modify the incumbent solution, and of completion algorithms that create feasible solutions from such neighbors. Also focusing on the improvement of a LNS framework, Mairy et al. [43] relied on the selection of parameters for the size of the operations that creates the neighborhood, the selection of the neighborhood operators, and the limit of an exploratory step that uses standard CP search algorithms for completion.

3.3 Case Discrimination Patterns

The reliability of an adaptive mechanism depends on the information to which it can refer to behave differently in each case. Such information is derived from the way in which the available data is used to learn and to outline the search. If obtained by sound learning methods, it creates theories regarding the choices that are made, which can be used to explain the observed performance variations between different search algorithms. The patterns listed hereafter illustrate the varied levels of detail in which available data can be used.

Atomic

Intent: The problem is considered as an atomic unit for improving performance. Peculiarities that can impact the choices being made on specific cases are ignored.

Application: The structure of the problem being solved is not well outlined, or not fully understood to leverage a successful adaptation.

References: In some cases, the atomic discrimination simply reflects the fact that this dimension of the search design was simply ignored. That is the case of many early approaches on the topic. However, Carchrae and Beck [11] opt consciously for this strategy. They praise it as a low-knowledge control of the search, which is aimed to facilitate the development of the solver by not requiring from the developer a deep knowledge about the domain of the problem.

General Profile

Intent: Each problem is depicted upon a standard profile. Search modifications are made according to the values of the properties of such profile.

Application: The problem can be described with a standard set of features, and their values vary enough for discriminating among the expected instances.

References: The standard profile of CSPs is usually based on properties extracted from binary versions of the models, i.e., the constraints are decomposed into pair-wise relations. Such type of approach can be found in the works of Gomes and Selman [22] as well as of Epstein et al. [16]. In the latter case, which is a part of the ACE project, the combination of such properties is considered as a footprint of the problem being solved. Such properties are used by Arbelaez et al. [3] to perform a fine-grained selection of branching heuristics. A vector of features is also used by Yun and Epstein [66] to group similar problems.

Domain-Specific

Intent: The depiction of the problem is made upon a specialized profile, which is made of specific properties from the domain of application of each problem.

Application: The use of application-related features allows a more precise and comprehensive instance characterization, thus providing a better classification.

References: Kadioglu et al. [35] described varied sets of features aimed to guide the selection of algorithms on different types of problems.

The idea of modifying the search with inference rules on problem constraints is mentioned as future work for the Multi-TAC solver [44]. In some CBR approaches [41, 20], the cases are based on the presence of constraints of a specific type of problem. The presence of specific constraints is also used by Hentenryck and Michel [26] to guide the synthesis of local search algorithms. In AEON [45], global constraints for scheduling are used to classify the problems.

An alternative path is followed in the QFP approach [8]: mechanisms for detecting thrashing in the search are used for triggering search modifications. Since the design of such mechanisms is strongly dependent of the kind of algorithm and problem, they were considered as an example of this pattern.

3.4 Aims and Limitations of the Current Depiction

The description above aimed to dissect existing design possibilities of adaptive search methods for CP in three dimensions. A number of patterns were described to meet the needs of varied cases with different possibilities regarding when the learning process occurs, which type of choice is made to modify the search behavior, and what sort of information is used to discriminate among different combinations of state and result. Beyond the fact that a number of other types of patterns can be conceived to complement those, the review of standard references upon the intended characterization covered mostly approaches to the CSP. There are notable approaches to similar problems, as is the case of the Satisfiability problem (SAT), which would require a separate work to be properly addressed. In addition, an attempt was made to restrict the description of each approach to the types of patterns that more closely relate to the issues it aimed to address. As a consequence, it can be observed that alternatives of search modification were discussed more extensively than other types of patterns. Therefore, the intended depiction here present is meant as a CSP-focused effort to complement recent contributions of general purpose on the topic. They include the discussion raised by Hamadi et al. [25], the revisional effort performed by Smith-Miles [55], and the generalization of solving methods developed by Battiti et al. [5].

4 Where Are We and What's Next?

The review of existing work on adaptive search methods evidences that there is a large number of successful approaches, but also that researchers have diverse and often conflicting views on how to succeed with their development. The use of a framework based on design patterns was intended to represent such diversity as the solution to different problems of autonomous search design, what could be easily assimilated by software developers and industry practitioners intending to make the most of CP. Whether the community decides to adopt the characterization as depicted here or not, the agreement on any comprehensive perspective on the topic would be instrumental to foster further development and specialization for varied purposes. A similar effort is observed in the work of Hooker [29] to develop an integrated perspective on optimization methods. Such perspective is based on the depiction of commonalities, differences and possibilities of integration among methods of global optimization, mathematical programming, constraint programming, and heuristics in general. The justification for both cases can be found in an essay authored by Hooker [28] as well, in which it is argued that CP and operations research in general are fields highly focused on the development of computational methods instead of explanatory models. Nevertheless, such models would provide a greater insight about the problems being approached and the methods underlying their solution. In the current case, we attempted to make the proposed model more accessible with the use of a standard outline that has been widely accepted by software engineers, thus helping to figure how to leverage machine learning techniques for constraint solving.

As such kind of explanatory model, the effort here developed still represents an early step aimed at achieving a pragmatic outlook of adaptivity for CP. If the depiction here present is to be adopted, at least three directions of debate and research are left as future work: (i) a broader discussion should be promoted with regard to the types and outline of patterns suggested, in resemblance to what happened in the software engineering community [19]; (ii) a comprehensive experimental analysis must be performed to evaluate in detail the use and mutual interaction of adaptive search patterns; and (iii) it might be the case that different types of solver competitions should be promoted as a means to evaluate varied conditions that may interest to real-world applications. In fact, the latter two directions would be beneficial if explored for any characterization resembling what was attempted in the current work.

Acknowledgements We are grateful for the comments and suggestions from A.A. Ciré, M.I. Harris, S. Harris, A.G.S. Mello, E.M. Nishida, T.F. Tavares, Y. Wakabayashi, some of the referenced authors, and the anonymous reviewers who evaluated preliminary versions of this work and contributed to its improvement.

References

1. Aldous, D., Vazirani, U.: "Go With the Winners" Algorithms. In: Proceedings of FOCS 1994, pp. 492–501, Santa Fe, USA (1994)
2. Arbelaez, A., Hamadi, Y.: Continuous Search in Constraint Programming: An Initial Investigation. In: Proceedings of CP 2009 Doctoral Program, Lisbon, Portugal (2009)
3. Arbelaez, A., Hamadi, Y., Sebag, M.: Online Heuristic Selection in Constraint Programming. In: Proceedings of SoCS'09, Lake Arrowhead, USA (2009)
4. Barták, R.: Theory and Practice of Constraint Propagation. In: Proceedings of CPDC 2001 Workshop, pp. 7–14, Gliwice, Poland (2001)
5. Battiti, R., Brunato, M., Mascia, F.: Reactive Search and Intelligent Optimization. Springer, New York, USA (2008)
6. Beek, P. van: Backtracking Search Algorithms. In: Rossi, F., Beek, P. van, Walsh, T. (eds.) Handbook of Constraint Programming. Elsevier, Amsterdam (2006)
7. Borret, J., Tsang, E. P. K.: Adaptive Constraint Satisfaction: The Quickest First Principle. Computational Intelligence, ISRL 1, pp. 203–230 (2009)
8. Borret, J., Tsang, E. P. K., Walsh, N. R.: Adaptive Constraint Satisfaction. In: Proceedings of the Workshop of the UK Planning and Scheduling Special Interest Group, Liverpool, UK (1996)
9. Boussemart, F., Hemery, F., Lecoutre, C., Sais, L.: Boosting Systematic Search by Weighting Constraints. In: Proceedings of ECAI 2004, pp. 146–150, Valencia, Spain (2004)
10. Cambazard, H., Jussien, N.: Identifying and Exploiting Problem Structures Using Explanation-based Constraint Programming. Constraints, 11(4), pp. 295–313 (2006)
11. Carchrae, T., Beck, J. C.: Applying Machine Learning to Low-Knowledge Control of Optimization Algorithms. Computational Intelligence, 21(4), pp. 372–387 (2005)
12. Corne, D., Reynolds, A.: Optimisation and Generalisation: Footprints in Instance Space. In: Proceedings of PPSN 2010, pp. 22–31, Krakow, Poland (2010)
13. Correia, M., Barahona, P.: On the Efficiency of Impact Based Heuristics. In: Proceedings of CP 2008, pp. 608–612, Sidney, Australia (2008)
14. Dechter, R.: Constraint Processing. Morgan Kaufmann, San Francisco, USA (2003)
15. El Sakkout, H., Wallace, M. G., Richards, E. B.: An Instance of Adaptive Constraint Propagation. In: Proceedings of CP 1996, pp.164–178, Cambridge, USA (1996)
16. Epstein, S. L., Freuder, E. C., Wallace, R.: Learning to Support Constraint Programmers. In: Computational Intelligence, 21(4), pp. 336–371 (2005)
17. Epstein, S. L., Freuder, E. C., Wallace, R., Morozov, A., Samuels, B.: The Adaptive Constraint Engine. In: Proceedings of CP 2002, pp. 525–542, Ithaca, USA (2002)
18. Gagliolo, M., Schmidhuber, J.: Learning Restart Strategies. In: Proceedings of IJCAI-07, Hyderabad, India (2007)
19. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Boston, USA (1995)

20. Gebruers, C., Hnich, B., Bridge, D., Freunder, E.: Using CBR to Select Solution Strategies in Constraint Programming. In: Proceedings of ICCBR 2005, pp. 222–236, Chicago, USA (2005)
21. Gomes, C. P., Fernández, C., Selman, B., Bessière, C.: Statistical Regimes Across Constrainedness Regions. In: Proceedings of CP 2004, pp. 32–46, Toronto, Canada (2004)
22. Gomes, C. P., Selman, B.: Algorithm Portfolios. Artificial Intelligence, 126, pp. 43–62 (2001)
23. Gomes, C. P., Selman, B.: The Science of Constraints. Constraint Programming Letters, 1, pp. 15–20 (2007)
24. Gomes, C. P., Selman, B., Kautz, H.: Boosting Combinatorial Search Through Randomization. In: Proceedings of AAAI 98, Madison, USA (1998)
25. Hamadi, Y., Monfroy, E., Saubion, F.: What is Autonomous Search? Technical Report MSR-TR-2008-80, Microsoft Research, Cambridge, UK (2008)
26. Hentenryck, P. V., Michel, L.: Synthesis of Constraint-Based Local Search Algorithms from High-Level Models. In: Proceedings of AAAI-07, Vancouver, Canada (2007)
27. Hentenryck, P. V., Perron, L., Puget, J. -F.: Search and Strategies in OPL. ACM Transactions on Computational Logic, 1(2) (2000)
28. Hooker, J. N.: Good and Bad Futures for Constraint Programming (and Operations Research). Constraint Programming Letters, 1, pp. 21–32 (2007)
29. Hooker, J. N.: Integrated Methods for Optimization. Springer, New York, USA (2007)
30. Hoos, H. H., Stützle, T.: Stochastic Local Search: Foundations and Applications. Morgan Kaufmann, San Francisco, USA (2004)
31. Huberman, B., Lukose, R., Hogg, T.: An Economics Approach to Hard Computational Problems. Science, 275, pp. 51–54 (1997)
32. Hulubei, T., O'Sullivan, B.: Search Heuristics and Heavy-Tailed Behaviour. In: Proceedings of CP 2005, pp. 328–342, Barcelona, Spain (2005)
33. Hutter, F., Hamadi, Y., Hoos, H. H., Leyton-Brown, K.: Performance Prediction and Automated Tuning of Randomized and Parametric Algorithms. In: Proceedings of CP 2006, pp. 213–228, Nantes, France (2006)
34. Kadioglu, S., Malitsky, Y., Sabharwal, A., Samulowitz, H., Sellmann, M.: Algorithm Selection and Scheduling. In: Proceedings of CP 2011, pp. 454–469, Perugia, Italy (2011)
35. Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K.: ISAC – Instance-Specific Algorithm Configuration. In: Proceedings of ECAI 2010, pp. 751–756, Lisbon, Portugal (2010)
36. Kadioglu, S., O'Mahony, E., Refalo, P., Sellmann, M.: Incorporating Variance in Impact-Based Search. In: Proceedings of CP 2011, pp. 470–477, Perugia, Italy (2011)
37. Kautz, H., Horvitz, E., Ruan, Y., Gomes, C., Selman, B.: Dynamic Restart Policies. In: Proceedings of AAAI-02, Edmonton, Canada (2002)
38. Laborie, P., Godard, D.: Self-Adapting Large Neighborhood Search: Application to Single-Mode Scheduling Problems. In: Proceedings of 3^{rd} MISTA, Paris, France (2007)
39. Leyton-Brown, K., Nudelman, G., Andrew, McFadden, J., Shoham, Y.: Boosting as a Metaphor for Algorithm Design. In: Proceedings of CP 2003, pp. 899–903, Kinsale, Ireland (2003)
40. Leyton-Brown, K., Nudelman, E., Shoham, Y.: Empirical Hardness Models: Methodology and a Case Study on Combinatorial Auctions. Jornal of the ACM, 56, 4, Article 22 (2009)

41. Little, J., Gebruers, C., Bridge, D., Freunder, E.: Capturing Constraint Programming Experience: A Case-Based Approach. In: International Workshop on Reformulating Constraint Satisfaction Problems, CP 2002, Ithaca, USA (2002)
42. Mackworth, A.: Consistency in Networks of Relations. Artificial Intelligence, 8(1), pp. 99–118 (1977)
43. Mairy, J.-B., Yves, D., Van Hentenryck, Pascal: Reinforced Adaptive Large Neighborhood Search. In: Proceedings of the 8^{th} LSCS, CP 2011, Perugia, Italy (2011)
44. Minton, S.: Automatically Configuring Constraint Satisfaction Programs: A Case Study. Constraints, 1(1), pp. 7–43 (1996)
45. Monette, J. -N., Deville, Y., Van Hentenryck, P.: AEON: Synthesizing Scheduling Algorithms from High-Level Models. In: Proceedings of 11^{th} INFORMS Computing Society Conference, Charleston, USA (2009)
46. O'Mahony, E., Hebrard, E., Holland, A., Nugent, C., O'Sullivan, B.: Using Case-based reasoning in an Algorithm Portfolio for Constraint Solving. In: Proceedings of AICS-2008, Cork, Ireland (2008)
47. O'Sullivan, B.: Automated Modelling and Solving in Constraint Programming. In: Proceedings of AAAI-10, Atlanta, USA (2010)
48. Petrovic, S. V., Epstein, S. L.: Tailoring a Mixture of Search Heuristics. Constraint Programming Letters, 4, pp. 15–38 (2009)
49. Puget, J. -F.: Constraint Programming Next Challenge: Simplicity of Use. In: Proceedings of CP 2004, pp. 5–8, Toronto, Canada (2004)
50. Refalo, P.: Impact-Based Search Strategies for Constraint Programming. In: Proceedings of CP 2004, pp. 557–571, Toronto, Canada (2004)
51. Rice, J. R.: The Algorithm Selection Problem - Abstract Models. Technical Report CSD-TR 116, Purdue University, West Lafayette, USA (1974)
52. Schulte, C., Stuckey, P. J.: Efficient Constraint Propagation Engines. ACM TOPLAS, 31, 1 (2008)
53. Selman, B., Mitchell, D. G., Levesque, H. J.: Generating Hard Satisfiability Problems. Artificial Intelligence, 81, pp. 17–29 (1996)
54. Serra, T.: Towards a Characterization of Adaptiveness for Constraint Programming Search Design In: CPAIOR 2011 Late Breaking Abstracts, pp. 36–37, Berlin, Germany (2011)
55. Smith-Miles, K. A.: Cross-Disciplinary Perspectives on Meta-Learning for Algorithm Selection. ACM Comput. Surv., 41, 1, Article 5 (2008)
56. Stamatatos, E., Stergiou, K.: Learning How to Propagate Using Random Probing. In: Proceedings of CPAIOR 2009, pp. 263–278, Pittsburgh, USA (2009)
57. Stergiou, K.: Heuristics for Dynamically Adapting Propagation. In: Proceedings of ECAI 2008, pp. 485–489, Patras, Greece (2008)
58. Streeter, M., Golovin, D., Smith, S. F.: Combining Multiple Heuristics Online. In: Proceedings of AAAI-07, Vancouver, Canada (2007)
59. Wallace, R. J.: Factor Analytic Studies of CSP Heuristics. In: Proceedings of CP 2005, pp. 712–726, Barcelona, Spain (2005)
60. Williams, R., Gomes, C., Selman, B.: Backdoors To Typical Case Complexity. In: Proceedings of IJCAI-03, Acapulco, Mexico (2003)
61. Wolpert, D. H., Macready, W. G.: No Free Lunch Theorems for Search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, Santa Fe, USA (1995)
62. Wolpert, D. H., Macready, W. G.: No Free Lunch Theorems for Optimization. IEEE Transactions on Evolutionary Computation, 1 (1997)
63. Wu, H., Beek, P. van: On Portfolios for Backtracking Search in the Presence of Deadlines. In: Proceedings of ICTAI-2007, pp. 231–238, Patras, Greece (2007)

64. Xu, L., Hoos, H. H., Leyton-Brown, K.: Hydra: Automatically Configuring Algorithms for Portfolio-Based Selection. In: Proceedings of AAAI-10, Atlanta, USA (2010)
65. Xu, L., Hutter, F., Hoos, H. H., Leyton-Brown, K.: SATzilla-07: The Design and Analysis of an Algorithm Portfolio for SAT. In: Proceedings of CP 2007, pp. 712–727, Providence, USA (2007)
66. Yun, X., Epstein, S. L.: Learning Algorithm Portfolios for Parallel Execution. In: Proceedings of LION 6, Paris, France (2012)
67. Zanarini, A., Pesant, G.: Solution Counting Algorithms for Constraint-Centered Search Heuristics. In: Proceedings of CP 2007, pp. 743–757, Providence, USA (2007)