Extensible CP-Based Autonomous Search

Broderick Crawford[1,2], Ricardo Soto[1], Carlos Castro[2], and Eric Monfroy[2,3]

[1] Pontificia Universidad Católica de Valparaíso, Chile
[2] Universidad Técnica Federico Santa María, Chile
[3] CNRS, LINA, Université de Nantes, France
{broderick.crawford,ricardo.soto}@ucv.cl,
{carlos.castro,eric.monfroy}@inf.utfsm.cl

Abstract. A main concern in Constraint Programming (CP) is to determine good variable and value order heuristics. However, this is known to be quite difficult as the effects on the solving process are rarely predictable. A novel solution to handle this concern is called Autonomous Search (AS), which is a special feature allowing an automatic reconfiguration of the solving process when a poor performance is detected. In this paper, we present a preliminary architecture for performing AS in CP. The idea is to perform an "on the fly" replacement of bad-performing heuristics by more promising ones. Another interesting feature of this architecture is its extensibility. It is possible to easily upgrade their components in order to improve the AS mechanism.

Keywords: Constraint Programming, Autonomous Search, Heuristic Search.

1 Introduction

Constraint Programming (CP) is known to be an efficient technology for modeling and solving constraint-based problems. It has emerged as a combination of ideas from different domains such as operation research, artificial intelligence, and programming languages. Currently, CP is largely used in diverse application areas, i.e., scheduling, configuration, diagnosis, engineering design, games, and bioinformatics. In CP, problems are formulated as Constraint Satisfaction Problems (CSP), which are defined by a triple $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{X}$ is an n-tuple of variables $\mathcal{X} = \langle x_1, x_2, \ldots, x_n \rangle$. $\mathcal{D}$ is a corresponding n-tuple of domains $\mathcal{D} = \langle D_1, D_2, \ldots, D_n \rangle$ such that $x_i \in D_i$, and D_i is a set of values, for $i = 1, \ldots, n$. $\mathcal{C}$ is an m-tuple of constraints $\mathcal{C} = \langle C_1, C_2, \ldots, C_m \rangle$, and a constraint C_j is defined as a subset of the Cartesian product of domains $D_{j_1} \times \cdots \times D_{j_{n_j}}$, for $j = 1, \ldots, m$. A solution to a CSP is an assignment $\{x_1 \rightarrow a_1, \ldots, x_n \rightarrow a_n\}$ such that $a_i \in D_i$ for $i = 1, \ldots, n$ and $(a_{j_1}, \ldots, a_{j_{n_j}}) \in C_j$, for $j = 1, \ldots, m$.

The common approach for solving CSPs is to employ a backtracking-like algorithm that interleaves two main phases: enumeration and propagation. In the enumeration phase, a value is assigned to a variable in order to check whether this instantiation is a feasible solution. The propagation attempts to prune the search space by filtering from domains the values that do not lead to any solution. The enumeration phase involves two important decisions: the order in which the variables and values are selected. This selection refers to the variable and value ordering heuristics, and jointly constitutes the enumeration strategy. It is well-known that those decisions are dramatically important in the performance of the solving process. However, to perform an a priori decision is hard, since the effects of the strategy are normally unpredictable.

Autonomous Search (AS) [2] is a special feature allowing systems to improve their performance by self-adaptation. This approach can smartly be applied to CP in order to reconfigure bad-performing processes produced by the use of an inappropriate heuristic. AS has been successfully applied in different solving and optimization techniques [3]. Among others, an interesting application is about parameter setting in evolutionary computing [5]. In this context, there exists a theoretical framework as well as different successful implementations. Unlike evolutionary computing, the AS-CP couple is more recent. A few works have reported promising results based on a similar theoretical framework [1], but little work has been done in developing extensible architectures for AS in CP.

The main advantage of AS in CP is to allow a self-adaptation of the search process when a poor performance is detected. The idea is to reconfigure the process by replacing bad-performing heuristics by more promising ones. In this paper, we present a preliminary architecture for implementing AS in CP. This new framework performs the replacement by measuring the quality of strategies through a choice function. The choice function determines the performance of a given strategy in a given amount of time. It is computed based upon a set of indicators and control parameters which are carefully adjusted by an optimizer. An important capability of this new framework is the possibility of easily update its components. This is useful for experimentation tasks. Developers are able to add new choice functions, new control parameter optimizers, and/or new ordering heuristics in order to test new AS approaches.

This paper is organized as follows. Section 2 presents the basic notions of CSP solving. The AS-CP architecture is described in Section 3, followed by the conclusion and future work.

2 CSP Solving

As previously mentioned, the basic idea for solving CSPs is to combine two main phases: enumeration and propagation. A general procedure for solving CSPs is depicted below.

```
load_CSP()
while NOT all_variables_fixed OR failure
   heuristic_variable_selection()
   heuristic_value_selection()
   propagate()
   if empty_domain_in_future_variable()
      shallow_backtrack()
   if empty_domain_in_current_variable()
      backtrack()
end_while
```

The goal is to iteratively generate partial solutions, backtracking when an inconsistency is detected, until a result is reached. The algorithm begins by loading the CSP model. Then, a while loop encloses a set of actions to be performed until fixing all the variables (i.e. assigning a consistent value) or a failure is detected (i.e. no solution is found). The first two enclosed actions correspond to the variable and value selection. The third action is responsible for the propagation phase. Finally, two conditions are included to perform backtracks. A shallow backtrack corresponds to try the next value available from the domain of the current variable, and the backtracking returns to the most recently instantiated variable that still has values to reach a solution.

3 Architecture

Our framework is supported by the architecture proposed in [4]. This architecture consists in 4 components: SOLVE, OBSERVATION, ANALYSIS and UPDATE. SOLVE runs a generic CSP solving algorithm. The enumeration strategies used are taken from the quality rank, which is controlled by the UPDATE component. OBSERVATION takes snapshots in order to store the relevant information of the resolution process. The ANALYSIS component studies those snapshots, evaluates the different strategies, and provides indicators to the UPDATE component. Some indicators used are for instance, number of variables fixed by propagation, number of shallow backtracks, number of backtracks as well as the current depth in the search tree. The UPDATE component makes decisions using the choice function. The choice function determines the performance of a given strategy in a given amount of time. It is calculated based on the indicators given by the ANALYSIS component and a set of control parameters computed by an optimizer.

Figure 1 depicts a general schema of the framework. The UPDATE component has been designed as a plug-in for the framework in order to easily be upgraded.

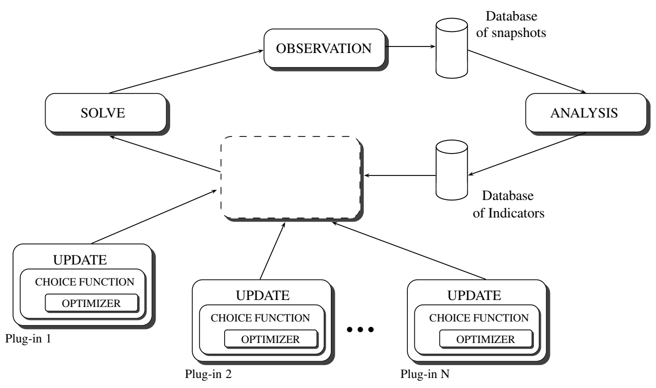

Fig. 1. Framework schema

Indeed, we have implemented a Java version of the UPDATE component which computes the choice function and optimizes its control parameters through a genetic algorithm. Another version of the UPDATE component, which is currently under implementation, uses a swarm optimizer.

3.1 The UPDATE Plug-In

In the current UPDATE component, we use a choice function [6] that ranks and chooses between different enumeration strategies at each step (a step is every time the solver is invoked to fix a variable by enumeration). For any enumeration strategy S_j, the choice function f in step n for S_j is defined by equation 1, where l is the number of indicators considered and α is the control parameter (it manages the relevance of the indicator within the choice function). Such control parameters are computed by the genetic algorithm, which attempt to find the values for which the backtracks are minimized.

$$f_n(S_j) = \sum_{i=1}^{l} \alpha_i f_{i_n}(S_j) \qquad (1)$$

Additionally, to control the relevance of an indicator i for an strategy S_j in a period of time, we use a popular statistical technique for producing smoothed time series called exponential smoothing. The idea is to associate, for some indicators, greater importance to recent performance by exponentially decreasing weights to older observations. In this way, recent observations give relatively more weight that older ones. The exponential smoothing is applied to the computation of $f_{i_n}(S_j)$, which is defined by equations 2 and 3, where v_0 is the value of the indicator i for the strategy S_j in time 1, n is a given step of the process, β is the smoothing factor, and $0 < \beta < 1$.

$$f_{i_1}(S_j) = v_0 \qquad (2) \qquad\qquad f_{i_n}(S_j) = v_{n-1} + \beta_i f_{i_{n-1}}(S_j) \qquad (3)$$

Let us note that the speed at which the older observations are smoothed (dampened) depends on β. When β is close to 0, dampening is quick and when it is close to 1, dampening is slow.

The general solving procedure including AS can be seen below.

```
load_CSP()
while NOT all_variables_fixed OR failure
   heuristic_variable_selection()
   heuristic_value_selection()
   propagate()
   if empty_domain_in_future_variable()
      shallow_backtrack()
   if empty_domain_in_current_variable()
      backtrack()
   calculate_indicators()
   calculate_choice_function()
   enum_strategy_selection()
end_while
```

Three new function calls have been included at the end: for calculating the indicators, the choice function, and for choosing promising strategies, that is, the ones with highest choice function. They are called after constraint propagation to compute the real effects of the strategy (some indicators may be impacted by the propagation).

4 Conclusion and Future Work

In this work, we have presented an extensible architecture for performing AS in CP. It allows the system self-adaptation by performing an automatic replacement of bad-performing heuristics. A main feature is perhaps the possibility of upgrading the UPDATE component. This allows users to modify or replace the choice function and/or the optimizer in order to perform new AS-CP experiments. The framework has been tested with different instances of several CP-benchmarks (send+more=money, N-queens, N-linear equations, self referential quiz, magic squares, sudoku, knight tour problem, etc) by using the already presented UPDATE component. A clear direction for future work is about adding new UPDATE components. This may involve to implement new optimizers as well as the study of new statistical methods for improving the choice function.

References

1. Crawford, B., Montecinos, M., Castro, C., Monfroy, E.: A hyperheuristic approach to select enumeration strategies in constraint programming. In: Proceedings of ACT 2009, pp. 265–267. IEEE Computer Society, Los Alamitos (2009)
2. Hamadi, Y., Monfroy, E., Saubion, F.: Special issue on autonomous search. Contraint Programming Letters 4 (2008)
3. Hamadi, Y., Monfroy, E., Saubion, F.: What is autonomous search? Technical Report MSR-TR-2008-80, Microsoft Research (2008)
4. Monfroy, E., Castro, C., Crawford, B.: Adaptive enumeration strategies and metabacktracks for constraint solving. In: Yakhno, T., Neuhold, E.J. (eds.) ADVIS 2006. LNCS, vol. 4243, pp. 354–363. Springer, Heidelberg (2006)
5. Robet, J., Lardeux, F., Saubion, F.: Autonomous control approach for local search. In: Stützle, T., Birattari, M., Hoos, H.H. (eds.) SLS 2009. LNCS, vol. 5752, pp. 130–134. Springer, Heidelberg (2009)
6. Soubeiga, E.: Development and Application of Hyperheuristics to Personnel Scheduling. PhD thesis, University of Nottingham School of Computer Science (2009)