Interactivity in Constraint Programming

Tomáš Müller

Charles University
Department of Theoretical Computer Science
Malostranske namesti 2/25, Prague, Czech Republic
muller@kti.mff.cuni.cz

Abstract. In the current computing environment, the interactive behaviour of software is a significant feature requested by many users. The minimal perturbation problem is often highly demanded and, in a sense, it seems to be very close to the interactivity. In this paper we want to compare these two features and also summarize their main demands and impacts on the constraint satisfaction problem. We also suggest several improvements toward these features.

Introduction

A finite constraint satisfaction problem (CSP) consists of a set of variables associated with finite domains and a set of constraints restricting the values that the variables can simultaneously take. A solution to a CSP is an assignment of a value from a particular domain to every variable, in such a way that every constraint is satisfied [8].

During the last few years, thanks to the faster, smaller, cheaper personal computers and well-developed, user friendly software, the interactive behaviour is more and more requested. The question is, what does this very popular interactivity exactly mean? And, what demands does it make to the current constraint solvers? In this paper, we will try to focus on the following issues.

Interactivity manifests itself in several ways:

At first, the possibility to influence the problem solving process seems to be the most important. On one hand, this capability can give the users a very powerful tool to tackle with difficulties that are sometimes very hard to solve for an automated solver. In such case, the user can guide the solver to the right direction, to search the right part of the solution space. On the other hand, the users very often do not even want to have a fully automated software. They prefer to have their piece of control, because they are responsible for the results and they also often do not believe that the software can find something reasonable or what they exactly want. In all these cases, it is much easier to convince the users to use the software when it does not act as a black box. Sometimes, there can also be plenty of constraints, exceptions or preferences which are very hard to express in some exact, computer understandable language.

When the user can influence the system, the solver has to be able to handle such modified solution and to continue solving the problem. And it also has to take into account these user changes, to prioritise them towards the original demands and expressed constraints. The generated solution should not differ much from the previous one (what the user modified), because the user has to be able to follow up the changes. If the solver gives a completely different solution, the system will be useless.

Another, also very important property is the visualisation of results. Not only some feasible, final solutions, but also some incomplete and even inconsistent solutions, generated during the search or when feasible results cannot be found in some reasonable time. It can be very hard and often very problem specific to present sufficiently the (sub-) results. The user should understand where the problems are and why a feasible or better solution was not found. What constraints are violated and why, how to help the system (e.g. by weakening some of the constraints, enlarging some of the domains etc.) to find a better solution. The possibility to observe what is happening inside, how the solution is being built, may also be very important (e.g. for debugging purposes, but the final users like the animation of the problem solving process as well).

The next very interesting feature, intimately connected with visualisation and the ability of influence the search process, is the capability to help the user make some decisions. The system can for example generate and evaluate some hints, like what to change and how (e.g. what constraint to weaken). There can also be more sophisticated advices, like what impact will some change have on the difference from the previous solution, what next changes will such modification cause, what has to be changed next.

In the following sections, we will consider (in a sense extreme) variants of the interactive behaviour in CSP. The first one (inspired by [3, 4, 5]) is the immediate response to a single user change. The system is requested to promptly incorporate the user’s modification, but in this case the search space is in most cases very small. We can for example highly limit the depth of the backtracking search or the number of iterations of some local search technique.

The second variant (inspired by [6]) is much more focused on the minimizing of the differences between the next generated solution and some previous one. We can consider working in some phases, where the solution is published between them. In this case we much more require minimization of the differences (e.g. number of variables changed) not to the previous generated solution but to the previous published one. On the other hand, we will not require immediate responses, but for example some slower calculation (not more than several minutes) after several changes. So, this case is much more focused to the minimal perturbation problem [7] than to the prompt response.

An Immediate Response Problem

Let’s focus on the first problem mentioned above. We have some initial variables, domains and constraints defined. We can also have an initial solution, probably made with some non-interactive automated solver. This solution can be either feasible or non-feasible, but in both cases, the user wants to change it. In this case the task is to immediately respond to these user changes. For example, when the user decides to remove some value from some variable's domain or to add a new constraint, the system should try to improve the changed solution, to find a better one applying some minimal amount of changes.

An interesting idea could be working with feasible solutions, where no hard constraint is violated, but where some of the variables are unassigned. Such solution can be given either via backtracking search (e.g. some not extensible, best ever found solution) or via some neighbour search technique (after un-assigning some values from the conflict variables). Such partial feasible solution can also be easier for presentation to the user than some infeasible one. We still need to represent soft conflicts, but it can be much easier than for example representing three lectures taking the same room at the same time. Now, let the user change something. He or she can add or remove a variable, add, remove, or alter a constraint. Visibly, the solution may become infeasible after such change. Nevertheless, there is a possibility to leave some variables unassigned, e.g. when there is no feasible solution by applying three or less operations.

What algorithm can provide us such behaviour? What about extension of some backtracking based algorithms? In this case, we can, for example, adopt variable and value selection heuristics as follows:

For the selection of variables, we will take into account only those variables which previous values are not assigned or not involved in some hard conflict with newly assigned variables. The variable selection criterion will therefore work with a set of variables that are possible to be selected. Initially, there will be all unassigned variables or only variables inconsistent with the user change. So, when the labelling instantiates a value to some variable, some other variables and their previously assigned values can become inconsistent with this assignment. These variables will be added into this set of variables and will be considered in the variable selection criterion for the next selection.

The value selection criterion may also be changed. When it seeks for the value of the selected variable, it should prefer not to violate previously selected values of the other non assigned variables. In other words, it should minimize the number of variables added to the set for the variable selection criterion. When there are several such values, usual best fit technique can be used (minimize soft conflicts and so on) [1]. The difficulty of selecting different values for the variables added to the variable selection set can also be taken into account.

Finally, the solver finishes when there are no variables for the variable selection criterion.

Because of the need of the prompt response, we also have to limit the solver somehow. There are several possibilities. We can, for example, limit the depth of the above described mutation of the backtracking search, but it seems to be much better to limit the number of changes according to some evaluation of differences between the generated solution and the previous solution. We can, for example, limit the weighted sum of the variables that were assigned different values from what they had before the solver was executed.

The found solution or other memorized affects from searching can also be used to help the user make some decisions afterwards. But what about some pre-processing of the previous solution, while the user is browsing it and searching for some improvement? For example, imagine an algorithm for playing chess: While the user is thinking how to move, the computer is searching for the response (and gradually increasing the depth of the search space). Can we do something similar in the constraint satisfaction problems? What about generating some hints, like: what if this variable will come into a conflict, what another value from its domain should be selected? For every variable, we can search for some alternative value (different from the currently assigned one). If there is enough time, we can also start increasing the depth, to search for consequences of assigning of some alternative value of some variable.

Minimal Perturbation Problem

The second problem mentioned above is rather different. We are not seeking for the prompt response, but much more we tend to minimize the perturbations between the next solution and some previous one. Consider for example the following timetabling problem: At first, there is an initial timetable found and published. Next, the feedback from the involved people (workers, teachers, students, nurses, …) has to be taken into account and propagated into the following revised timetable. This can also takes several rounds of revisions. In each stage, we need to minimize the differences between the previous and the following published timetables, because we do not want to bother and/or involve other people, who possibly agreed with the previous published timetable.

This problem seems to be totally different from the previous one. But note that we again want to start from some (partially-) feasible solution with the given set of changes and we want to generate a solution with some minimised number of differences (according to some metrics). Also, the adaptation of the variable and value selection heuristics can be very similar: Variable selection criterion can start with the variables that cannot take their previous (published) values and the value selection criterion will again try to minimize further conflicts with previous values of the other not yet instantiated variables.

For the acceleration and precision of the selection heuristics, it seems to be very useful to pre-process the variables that cannot take their previous values with some local search or limited breadth-first or depth-first search. The output of such pre-processing can be twofold: it should give us information about the "best" alternative values for variables and it should also classify the variables according to the probability of assignment of their previous values. Such results can be used during the subsequent search (labelling of the variables). This information can also be generated for the variables, touched by the alternatives of the conflicting variables and so on, into some limited depth.

The pre-processing process described above can also be executed during the labelling (e.g. when some required information is missing). In such case, the pre-processing search can first try not to conflict already assigned variables. It can also return some information about how far we need to backtrack from some dead-end node of the search tree.

Conclusion

We presented some general motivations, ideas, and directions in the field of interactivity in the constraint satisfaction problem and its applications. We also proposed some algorithms and ideas how to solve these tasks via improving variable and value selection criteria. We believe that these motivations and ideas are successfully applicable to many constraint satisfaction problems.

Current research could lead to the development, improvement, implementation, and comparison of the above described algorithms.

References

[1] D. Clements, J. Crawford, D. Joslin, G. Nemhauser, M. Puttlitz, M. Savelsbergh. *Heuristic optimization: A hybrid AI/OR approach.* In Workshop on Industrial Constraint-Directed Scheduling, 1997.

[2] Z. Michalewicz, D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer, 2000.

[3] T. Müller and R. Barták. *Interactive Timetabling*. In Proceedings of the ERCIM Workshop on Constraints, Prague, 2001.

[4] T. Müller and R. Barták. *Interactive Timetabling: Concepts, Techniques, and Practical Results*. In Proceedings to PATAT'02 Conference, Gent, 2002.

[5] T. Müller. *Interactive Heuristic Search Algorithm*. In Proceedings of the CP'02 Conference – Doctoral Programme, Ithaca, 2002

[6] H. Rudová, K. Murray, *University Course Timetabling with Soft Constraints.* In Proceedings to PATAT'02, Gent, 2002

[7] El. Sakkout and M Wallace. *Probe Backtrack Search for Minimal Perturbation in Dynamic Scheduling*, Constraints, Special Issue on Industrial Constraint-Directed Scheduling, Vol 5 No 4, 2000

[8] E. Tsang. Foundations of Constraint Satisfaction. Academic Press, 1993