LSVF: a New Search Heuristic to Reduce the Backtracking Calls for Solving Constraint Satisfaction Problem

Cleyton Rodrigues
Center of Informatics,
Federal University of
Pernambuco (CIn-UFPE)
Recife, PE, Brazil, FaculdadeEscritor
Osman da Costa Lins,
Vitória de Santo Antão – PE, Brazil

Ryan Ribeiro de Azevedo
Center of Informatics,
Federal University of
Pernambuco (CIn-UFPE)
Recife, PE, Brazil, Federal
University of Piauí (DSI-UFPI)
Caixa Postal 15.064 – 91.501-970 –
Picos – PI – Brazil

Fred Freitas, Eric Dantas
Center of Informatics,
Federal University of Pernambuco
(CIn-UFPE)
Recife, PE, Brazil

***Abstract*—Many researchers in Artificial Intelligence seek for new algorithms to reduce the amount of memory/ time consumed for general searches in Constraint Satisfaction Problems. These improvements are accomplished by the use of heuristics which either prune useless tree search branches or even indicate the path to reach the (optimal) solution faster than the blind version of the search. Many heuristics were proposed in the literature, like the Least Constraining Value (LCV). In this paper we propose a new pre-processing search heuristic to reduce the amount of backtracking calls, namely the Least Suggested Value First: a solution whenever the LCV solely cannot measure how much a value is constrained. In this paper, we present a pedagogical example, as well as the preliminary results.**

Keywords-Backtracking Call; Constraint Satisfaction Problems; Heuristic Search.

I. INTRODUCTION

Constraint Satisfaction Problems (CSP) still remains as a relevant Artificial Intelligence (AI) research field. Having a wide range of applicability, such as planning, resource allocation, traffic air routing, scheduling [Brailsford et al, 1998], CSP has been largely used for real large complex applications.

A tough problem that hampers its usage in a larger scale resides in the fact that, in general, CSP are NP-complete and combinatorial by nature. Amongst the various methods developed to handle this sort of problems, in this paper, our focus concerns the search tree approach coupled with the backtracking operation.

In particular, we address some of the several heuristics used so far to reduce (without guarantees) the amount of time needed to find a solution, namely: Static/ Dynamic Highest Degree heuristic (SHD/DHD), Most Constraint Variable (MCV) and Least Constraining Value (LCV) [Russell and Norvig, 2003]. Some problems, however, like the ones common referred as instances of the Four Colour Map Theorem [Robertson et al., 1997], present the same domain for each entity, making the LCV heuristic impossible to decide the best value to be asserted first. For these cases, we propose a new pre-processing heuristic, namely Least Suggested Value First (LSVF), which can bring significant gains by a simple domain value sorting, respecting an order made by the following question "Which is the least used value to be suggested now?". Additionally, we enumerate some assumptions to improve the ordering. Along the paper, we show some preliminary results with remarkable reduce of backtracking calls.

This paper is organized as follows. Section 2 explains briefly the formal definition of CSP and the most common heuristics used in this class of problems; following, Section 3 details the language CHR^V and why we have chosen it; Section 4 introduces the LSVF heuristic with a pedagogical example; a brief comparison between LCV and LSVF is performed in Section 5, showing that the heuristics are feasible in different scenarios, but exemplifying as LSVF can serve as a tie breaker for the LCV; Section 6 highlights some results, and finally, Section 7 presents the final remakes and the future works.

II. CSP AND HEURISTICS

In this section, we introduce the basic concepts of CSP and further, we detail the most common heuristics used for this kind of problem.

A. Constraint Satisfaction Problem

Roughly speaking, CSP are problems defined by a set of variables $X = \{X_1, X_2,...,X_n\}$, where each one (X_i) ranges in a known domain (D), and a set of Constraints $C = \{C_1, C_2,..., C_n\}$ which restricts specifically one or a group of variables with the values they can assume. A consistent complete solution corresponds to a full variable valuation, which is further in accordance with the constraints imposed. Along the paper, we refer to the variables as entities. Figure 1 depicts a pedagogical problem.

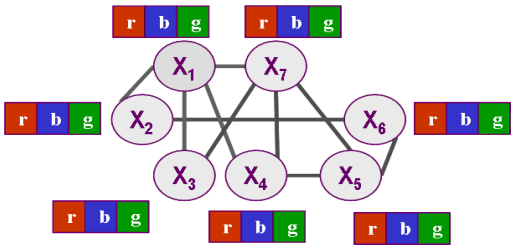

Figure 1. A Pedagogical Constraint Satisfaction Problem

In the figure above, the entities are the set $\{X_1, X_2, X_3, X_4, X_5, X_6, X_7\}$ and each one can assume one of the following value of the domain: D = {r,g,b}, referring to the colours, red, green, and blue, respectively. The only constraint imposed restricts the neighbouring places (that is, each pair of nodes linked by an arc) to have different colours. As usual, this problem can be reformulated into a search tree problem, where the branches represent all the possible paths to a consistent solution.

By definition, each branch not in accordance with C, must be pruned. The backtracking algorithm, a special case of depth-first, is neither complete nor optimal, in case of infinite branches [Vilain et al., 1990]. As we have not established an optimal solution to the problem, our worries rely only upon the completeness of the algorithm. However, we only take into account problems in which search does not lead to infinite branches, and thus, the completeness of the problem is ensured.

B. Search Heuristics

Basically, the backtracking search is used for this sort of problems. Roughly, in a depth-first manner, a value from the domain is assigned, and whenever an inconsistency is detected, the algorithm backtracks to choose another colour (another resource), if any is available. Although simple in conception, the search is far from being efficient. Moreover, this algorithm lacks intelligence, in the sense to re-compute partial valuations already proven to be consistent.

A blind search, like the backtracking, is improved in efficiency employing some heuristics. Regarding CSP, general heuristics (that is, problem-independent, opposite to domain-specific heuristics, as the ones in A* search [NationMaster, 2010]) methods speed up the search while removing some sources of random choice, as: "Which next unassigned variable should be taken?", "Which next value should be assigned?". The answer for the questions arises by a variable and value ordering. The most famous heuristics for variable and value ordering are highlighted below. Note that the two former methods concern the variable choice, and the latter refers to the value ordering:

- Most Constrained Variable (MCV) avoids useless computations when an assignment will eventually lead the search to an inconsistent valuation. The idea is to try first the variables more prone to causing errors;
- When the later heuristics is useless, the Degree Heuristic (SHD/DHD) serves as a tiebreaker for MCV, once it calculates the degree (number of conflicts) of each entity;
- The Least Constraining Value (LCV), in turn, sorts decreasingly the values in a domain respecting how much the value conflicts with the related entities (that is, the values less shared are tried first).

We have restricted our scope of research to the class of problems similar to the family of the four colours theorem, where the domain is the same for each entity. In this sense, the LCV heuristic is pointless since the level of constraining for each value is the same. This drawback forces us to search alternatives to sort the values of CSP in similar situations, but without sacrificing efficiency.

In the next section we describe CHR^v, a Constraint Logic Programming Language which we have used to carry out the tests. The language is built on Prolog, and its syntax/semantics allows structure CSP problems in a simple and clear manner.

III. CHR^V

Constraint Handling Rules with Disjunction (CHRv) [Abdennadher and Schutz, 1998] is a general concurrent logic programming language, rule-based, which have been adapted to a wide set of applications such as: constraint satisfaction [Wolf, 2005], abduction [Gavanelli et al, 2008], component development engineering [Fages et al, 2008], and so on. It is designed for creation of constraint solvers. CHR^v is a fully accepted logic programming language, since it subsumes the main types of reasoning systems [Frühwirth, 2008]: the production system, the term rewriting system, besides Prolog rules. Additionally, the language is syntactically and semantically well defined [Abdennadher and Schutz, 1998]. Concerning the syntax, a CHRV program is a set of rules defined as:

$$rule\_name@Hk \setminus Hr \Leftrightarrow G \mid B. \qquad (1.1)$$

Rule_name is the non-compulsory name of the rule. The head is defined by the user defined constraints represented by Hk and Hr, with which an engine tries to match with the constraints in the store. Further, G stands for the set of guard built in (native) constraints (available by the engine), that is, a condition imposed to be verified to fire any rule. Finally, B is the disjunctive body, corresponding to a set of constraints added within the store, whenever the rule fires. The logical conjunction and disjunction of constraints are syntactically expressed by the symbols "," and ";" respectively. Logically, the interpretation of the rule is as follows:

$$\forall V_{GH}(G \rightarrow ((H_k \wedge H_r) \leftrightarrow (\exists V_{B\setminus GH} B \wedge H_k))), \text{ where } V_{GH} = vars(G) \cup vars(H_k) \cup vars(H_r),\ V_{B\setminus GH} = vars(B) \setminus V_{GH} \qquad (1.2)$$

As the guard (G) of the rule consistent and true from the facts present, the user-defined constraints representend by Hk and Hr, are logically equivalent to the body (B) and Hk conjoined, so they can be replaced. This represents a Sympagation rule and the idea is to simplify the basis of facts to which the deductions can be made. We ask the reader to check the bibliography for further reference to the declarative semantics [Abdennadher and Schutz, 1998].

In the literature, many operational semantics was proposed, as [Abdennadher et al, 1999]. However, the ones most used in CHR^v implementations are based on the refined semantics [Duck et al, 2004] (as the SWI-Prologversion 5.6.52 [Wielemaker, 2008] used in the examples carried out along this paper). According the refined operational semantics, when more than one rule is possible to fire, it takes into account the order in which the rules were written in a program. Hence, as SHD heuristic orders the entities to be valued in accordance with the level of constraining, this pre-analysis help us to write the rules based on this sort. Thus, we could concentrate our effort on the order of the values in the domain.

The problem depicted in Figure 1 is represented by the logical conjunction of the following rules:

```
f@ facts ==> m, d(x1,C1), d(x7,C7), d(x4,C4),
d(x3,C3), d(x2,C2),d(x5,C5), d(x6,C6).
d1@ d(x1,C) ==> C=red; C=green; C=blue.
d7@ d(x7,C) ==> C=red; C=green; C=blue.
m@ m <=> n(x1,x2), n(x1,x3), n(x1,x4),
n(x1,x7), n(x2,x6),n(x3,x7), n(x4,x7),
n(x4,x5), n(x5,x7), n(x5,x6).
n1@ n(Ri,Rj), d(Ri,Ci), d(Rj,Cj)<=> Ci=Cj |
fail.
```

The first rule f@ introduces the constraints into the store, which is a set of predicates with functor d and two arguments: the entity and a variable to store the valuation of the entity. The seven following rules relate the entity with the respective domain. Additionally, rule m adds all the conceptual constraints, in the following sense: n(Ri,Rj) means there is an arc linking Ri to Rj, thus, both entities could not share the same colour. Finally, the last rule is a sort of integrity constraint. It fires whenever the constraints imposed is violated. Logically, it says that if two linked entities n(Ri,Rj) share the same colour (condition ensured by the guard), then the engine needs to backtrack to a new (consistent) valuation.

IV. Least Suggested Value First (LSVF)

Some points need be discussed to clarify the technique developed to improve the search, decreasing the amount of backtracking calls. The first point, which rule will trigger, was discussed before. The second important subject of discussion is the order of which the values are taken from the domain in the search.

We have already said that the logical disjunction is denoted in the body of a CHR^v rule, syntactically expressed as ";". In order to maintain consistency with the declarative semantics, CHR^{v}engine tries all the alternatives of a disjunctive body. A disjunctive body is always evaluated from left-to-right.

Taking the rule d1 from the previous example, the engine tries the following order for X_1: (1) red, (2) green and, (3) blue. All the rules were created respecting the same values' order. At first glance, we realized a relevant problem: if all entities try first the same colour, and we know that these entities are related, a second evaluated entity always needs to backtrack. Furthermore, since the entities shares the same domain, LCV is pointless: each value has the same level of constraining. In order to make our idea clear, we introduce a second example (Figure 2).

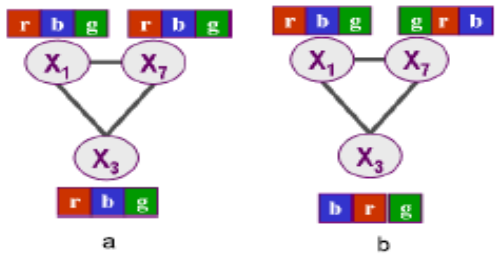

Figure 2. An example regarding the order of the colours.

The Figure 2a shows the motivation problem for the new heuristics discussed. There are 3 entities X_1, X_3, X_7, each one sharing the same domain. Let us respect the order of valuation from left to right, and the order of variable chosen based on the numerical order. Thus, the engine works as follows:

1) X_1 is chosen, and the colour red is taken;
2) X_3 is chosen, and the colour red is taken;
3) Inconsistency found: backtracking;
4) X_3 is chosen, and the colour blue is taken;
5) X_7 is chosen, and the colour red is taken;
6) Inconsistency found: backtracking;
7) X_7 is chosen, and the colour blue is taken;
8) Inconsistency found: backtracking;
9) X_7 is chosen, and the green is taken.

Following, in the Figure 2b, the values order is changed to avoid, as much as possible, the conflicts. The engine now works as stated below:

1) X_1 is chosen, and the colour red is taken;
2) X_3 is chosen, and the colour blue is taken;
3) X_7 is chosen, and the colour green is taken.

The above modification prevented the backtracking calls, and the solution was reached just with three steps, unlike the last example, which realized the same, in 9 steps. Evidently, in practice, we cannot avoid all backtracking calls, but each reduction is well-suited for the overall search time-consumption.

A. *How The Heuristics Works?*

Our propose is to enjoy the operational semantics addressed by the CHR^V implementation to sort the order in which the values from the domain is asserted, removing the amount of backtracking calls. We believe this reduction can fit well to large and complex problems, where time is a relevant factor.

The focus addressed by this paper is for problems with three or four elements in the domain. In this context, the entity set members are categorized as: (i) Soft Entities, that is, the less constrained ones, (ii) Middle Entities, which are half constrained, (iii) Hard Entities, which are, more constrained. The creation of these three groups is explained in the next subsection. Hence, instead of proposing a solution of random sorting, we have taken the following assumptions:

- Usually, the less constrained entities are likely to be linked to others more constrained, and, further, the entities less restricted are not connected to each other (if this were the case, the entities owned other

restrictions than those that connect them, and they would be deemed more constrained). Thus, the domain of these entities is sorted in the same manner;

- Normally, hard entities are linked to middle ones, and thus the order of valuation must be in conformance to this fact, example, if a hard entity domain is ordered like (1) red, (2) green, (3) blue, the middle should be sorted like (1) blue, (2) green (3) red, that is, the less suggested values first;
- The first value assumed by the hard entities should be the last for the soft and middle entities, since potentially both are linked to the former (this is why they were classified as hard).

B. Formalizing LSVF

After the explanation of how the heuristic works, it is important to define the levels of constraints (soft, middle, hard). This requires calculating the level restriction for each entity, provided by the heuristic SHD. Through this, it suffices for each element domain of each entity to calculate how many inconsistencies exist with respect to that element for its related entities. Formally, we define R as the function that takes an element of the domain (X_i) and returns the level of restriction (IN). The restriction level of an entity (e) as a whole, in turn, is defined as the sum of the return R for each domain element of this entity.

$$R : X_i \rightarrow IN$$

$$\text{level of restriction}(e) = \sum_{i=1}^{n} R(X_i) \qquad (1.3)$$

In order to divide the entities into the three groups, we just take the value of the most restricted entity and divide by three. With the quotient of dividing (Q), one should take the following classification:

- Soft Entities: Those whose level of restriction is near the value of Q;
- Middle Entities: Those whose level of restriction is near the value of 2Q;
- Hard Entities: Those whose level of restriction is near the value of 3Q;

As an example, suppose that for an arbitrary problem, the highest amount of restriction for an entity was 50. The quotient of the division by 3 is about 17. Thus, those entities whose restriction value is around 17 (Q) will be classified as soft; those whose value is around 34 (2Q) are classified as middle, and those with a value close to 51 (3Q) will be hard entities.

V. Experiments And Results

In order to exemplify this approach, we are going to show the reformulation of the example used along this paper, illustrating gradually the gains obtained. With respect the problem, we divided the set of entities as follows: (i) soft entities: {X_2, X_3, X_6}, (ii) middle entities: {X_4, X_5}, and (iii) hard entities {X_1, X_7}, with 6, 9 and 12 conflicts, respectively. Note that 12:3 = 4, then we have Q = 4, 2Q = 8, 3Q = 12. Table 1 summarizes the amount of inferences made and the number of backtracking calls. Inference represents the amount of deductions made by Prolog engine along a query, its amount is directly related to the time that a query was held, so the lower the number of inferences, the less time spent.

TABLE I. First Results with the LSVF Heuristic.

Sorting	Inferences	Backtracking
soft (r,g,b), middle (r,g,b), hard (r,g,b)	4,897	8
soft (r,g,b), middle (b,r,g), hard (r,g,b)	4,694	7
soft (g,r,b), middle (b,r,g), hard (r,g,b)	4,415	6
soft (g,b,r), middle (b,g,r), hard (r,g,b)	4,208	5

Not accidentally, the table was populated according to the assumptions raised earlier. Each line in the table corresponds to a different CHR^v program. In the first line, the heuristic was not used. It is worth to keep their results in the table to compare with the other levels, where the assumptions (which define the LSVF) were gradually applied. The second line has changed the first suggested colour of the Middle entities with respect the hard. Following, the third one has changed the first colour of domain of soft entities with respect the others (middle and hard).

There has been a reduction of 25% of backtrack calls in accordance with the first program. Finally, the last line has used all assumptions talked, and both measures were visibly reduced. In this latter case, the engine backtracks 5 times, three calls less than the original program. Note that the last program follows all the assumptions discussed, and the results obtained were remarkable. Before concluding the section, the paper further explores the new heuristic with larger problems.

To this end, we chose the map of Brazil to investigate the assumptions by checking, in parallel, the reduction in the amount of inferences and backtracking calls. Brazil is divided into 26 states and one federal unit, totalling 27 entities. As discussed previously, the idea is to colour these entities using three colours (red, green, blue), so that neighbouring regions do not have the same colours. Figure 3 shows the map as well as neighbouring states. According to the theorem of the four colours, two regions are called adjacent only if they share a border segment, not just a point. In the figure, the states that share a single point are connected by a shaded line. The programs can be found at http://cin.ufpe.br/~cmor/IBERAMIA/.

As before, the entities were divided into three types. The problem was analysed from three perspectives. At first, the domain of entities remained the same for everyone. With 74.553 inferences and 50 backtracking calls, a solution was reached. Then in the second perspective, the domain of middle entities was changed, while in the third and final perspective, beyond the middle, the domain of soft entities has been re-arranged. While in the second case, we obtained 71.558 inferences and 46 backtracking calls, the last, were 61.772 and 38, respectively.

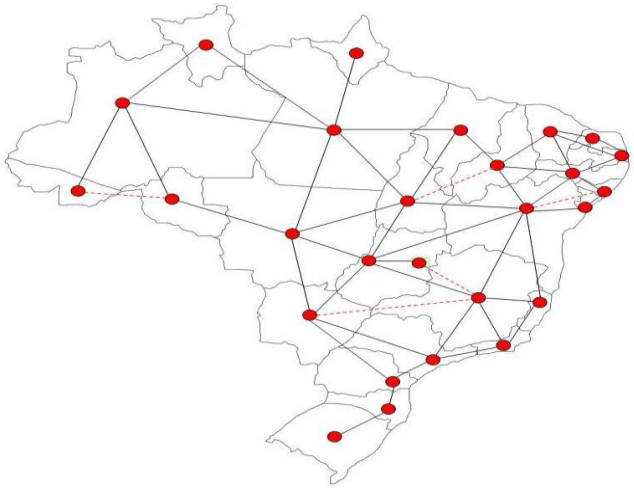

Figure 3. Map Colour of Brazil

Finally, to analyse the decline of these variables discussed so far, through a graph (Figure 4), we analysed 10 instances of colouring problems. Each instance has a multiple of six entities, starting with 6 and ending at 60. It can be observed by the first graphic (problem x amount of inferences) by using LSVF (W/LSVF) the curve is always kept lower than the curve without the heuristics (Wout/LSVF).

By analysing the problem by the amount of backtracking calls (graphic 2) the difference becomes deeper; since the W/LSVF curve follows a growth rate well below that the curve without the heuristic. As an example, the last problem (m10) with 60 entities, there is a decrease from 45 (no heuristics) to 5 (with heuristics) backtracking calls.

VI. LSVF As A Tie-Breaker For LCV

It is worth to say, most importantly, LCV and LSVF cannot be compared because they are used in different scenarios: while the former is used when the domain of the elements are different, the second, by contrast, is used when the domains are equal, leading to a situation impossible to sort the values using the LCV. However, it was observed that LSVF can be used in conjunction with LCV as a strategy to tie-break, even when the domains are not completely different.

Take the same example addressed in figure 1, but now, taking into consideration the following domains of variables: X_1 = {red, blue, green}, X_2 = {red, blue}, X_3 = {red, blue}, X_4 = {red, blue, green}, X_5 = {red, blue, green}, X_6 = {red, blue}, X_7 = {red, blue, green}.

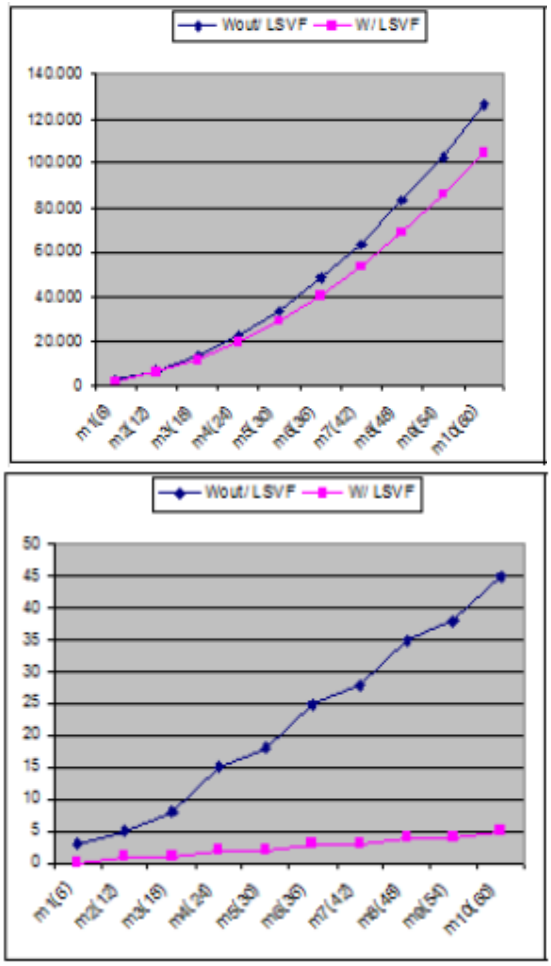

Figure 4. Results: Problem x Inference, Problem x Backtracking Calls.

Again, using the heuristic SHD, we calculate the conflicts of each variable (X_1=10, X_2=4, X_3=4, X_4=9, X_5=8, X_6=4, X_7=11) and, as before, we split into three groups: Hard {X_1, X_7} (entities with more conflicts), Middle {X_4, X_5} (entities with an average amount of conflict), Soft {X_2, X_3, X_6} (less conflicts). Moreover, the order of the values within each domain was defined based on the LCV heuristic. The table 2 summarizes the results (it was used only the initials of the colours).

Only with LCV (column 2), there were 4.210 inferences and 5 backtracking calls to reach a complete and consistent valuation. However, it was observed that for all entities, the constraining degree value between the colours blue and red was the same. By observation, and the assumption that soft entities are potentially linked to middle or hard ones, and except for the colour green (not possessed by soft entities), the order of values is the same, in column 3 (LCV + LSVF'), the values of soft entities domain were in inverted position. With this change, the number of inferences and backtracking calls was reduced to 4.024 and 4, respectively.

Finally, we noticed that the three colours for X_4 had the same level of restriction. Based on the assumption of the reverse order of values between Middle and Hard entities, in column 4 (LCV + LSVF") the domain of X_4 was re-arranged

as shown. In this case, there were 3.576 inferences and only 2 backtracking calls.

TABLE II. FIRST RESULTS WITH THE LSVF HEURISTIC.

Variable	LCV	LCV + LSVF'	LCV + LSVF''
X1	g, r, b	g, r, b	g, r, b
X7	g, r, b	g, r, b	g, r, b
X4	g, r, b	g, r, b	b, r, g
X5	g, r, b	g, r, b	g, r, b
X2	r, b	b, r	b, r
X3	r, b	b, r	b, r
X6	r, b	b, r	b, r

VII. FINAL REMARKS AND FUTURE WORK

The preliminary results obtained were very satisfactory. We might see that, as we organize the values of the domain of the entities, gradually the search has been getting more efficient with respect to the number of inferences necessary to reach a solution. It was important to mention that we are neither worried with optimal solutions nor with all the solutions for the problem. We only focus on our overall effort to reach a solution.

In order to validate completely the LSVF heuristics, our next step is to analyse the approach with more complex problems.

Additionally, our aim is to check the time resource allocated for this kind of problem. In previous analysis, it was noted that the reduction in the amount of backtracking tends to reduce, directly, the time needed to find a solution. In fact, during the analysis that resulted in the graphic above, the time has decreased in the last instances. Another path to be further explored, is to define specifically, the partnership between LCV and LSVF, i.e., when the second heuristic can be used together with the first.

REFERENCES

[1] Abdennadher, S. and Schutz, H. (1998) Chrv: A flexible query language. In: In FQAS 98: Proceedings of the Third International Conference on Flexible Query Answering Systems, Springer-Verlag, 1–14.

[2] Abdennadher, S., Fruhwirth, T. and Meuss, H. (1999) Confluence and semantics of constraint simplification rules. Constraints 4(2),133–165.

[3] Brailsford, S., Potts, C. and Smith, B. (1998) "Constraint satisfaction problems: Algorithms and applications". Technical report, University of Southampton - Department of Accounting and Management Science.

[4] Duck, G.J., Stuckey, P., de la Banda, M.G. and Holzbaur, C. (2004) The refined operational semantics of constraint handling rules. In: ICLP'04: Proceedings of the 20th International Conference on Logic Programming, Springer Berlin / Heidelberg, 90–104.

[5] Fages, F., Rodrigues, C. and Martinez, T. (2008) Modular CHR with ask and tell. In: CHR '08: Proc. 5th Workshop on Constraint Handling Rules, (Linz, Austria) 95–110.

[6] Frühwirth, T. (2008) Welcome to constraint handling rules. 1–15.

[7] Gavanelli, M., Alberti, M. and Lamma, E.(2008) Integrating abduction and constraint optimization in constraint handling rules. In: Proceeding of the 2008 conference on ECAI 2008, Amsterdam, The Netherlands, The Netherlands, IOS Press, 903–904.

[8] NationMaster (2010): Encyclopedia-decidability.

[9] Robertson, N., Sanders, D., Seymour, P. and Thomas, R. (1997) "The four-colour theorem". J. Comb. Theory Ser. B 70(1) 2–44.

[10] Russell, S. and Norvig, P. (2003) "Constraints Satisfaction Problems" . In: Artificial Intelligence: A Modern Approach. 2nd edition edn. Prentice-Hall, Englewood Cliffs, NJ 143–144.

[11] Vilain, M., Kautz, H. and Van Beek, P. (1990) Constraint propagation algorithms for temporal reasoning: a revised report. (1990) 373–381.

[12] Wielemaker, J. (2008) SWI-Prolog 5.6 Reference Manual.

[13] Wolf, A. (2005) Intelligent search strategies based on adaptive constraint handling rules. Theory Pract. Log. Program. 5(4-5), 567–594.