On the Integration of Singleton Consistencies and Look-Ahead Heuristics

Marco Correia and Pedro Barahona

Centro de Inteligência Artificial, Departamento de Informática,
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
{mvc,pb}@di.fct.unl.pt

Abstract. The efficiency of complete solvers depends both on constraint propagation to narrow the domains and some form of complete search. Whereas constraint propagators should achieve a good trade-off between their complexity and the pruning that is obtained, search heuristics take decisions based on information about the state of the problem being solved. In general, these two components are independent and are indeed considered separately. A recent family of algorithms have been proposed to achieve a strong form of consistency called Singleton Consistency (SC). These algorithms perform a limited amount of search and propagation (lookahead) to remove inconsistent values from the variables domains, making SC costly to maintain. This paper follows from the observation that search states being explored while enforcing SC are an important source of information about the future search space which is being ignored. In this paper we discuss the integration of this look-ahead information into variable and value selection heuristics, and show that significant speedups are obtained in a number of standard benchmark problems.

1 Introduction

Complete constraint programming solvers have their efficiency dependent on two complementary components, propagation and search. Constraint propagation is a key component in constraint solving, eliminating values from the domains of the variables with polynomial (local) algorithms. The other component, search, aims at finding solutions in the remaining search space, and is usually driven by heuristics both for selecting the variable to enumerate and the value that is chosen first.

Typically, these components are independent. In particular, heuristics take into account some features of the remaining search space, and some structure of the problem to take decisions. Clearly, the more information there is, the more likely it is to get a good (informed) heuristics. Recently, a lot of attention has been given to a class of algorithms which analyse look-ahead what-if scenarios: what would happen if a variable x takes some value v? Such look-ahead analysis (typically done by subsequently maintaining arc or generalised arc consistency on the constraint network) may detect that value v is not part of any solution, and

eliminate it from the domain of variable x. This is the purpose of the different variants of Singleton Consistency (SC) [7,1,4,15].

In this paper we propose to go one step further of the above approaches. On the one hand, by recognising that SC propagation is not very cost-effective in general [17], we propose to restrict it to those variables more likely to be chosen by the variable selection heuristics. More specifically, we assume that there are often many variables that can be selected and for which no good criteria exists to discriminate them. This is the case with the first-fail (FF) heuristics, where often there are many variables with 2 values, all connected to the same number of other variables (as is the case with complete graphs). Hence the information gain obtained from SC propagation is used to break the ties between the pre-selected variables.

On the other hand, we attempt to better exploit the information made available by the lookahead procedure, and use it not only to filter values but also to guide search. The idea of exploiting look-ahead information is not new. However in the context of Constraint Programming, look-ahead information has not been fully integrated in subsequent variable and value selection heuristics (see section 5).

In this paper we thus investigate the possibility of integrating Singleton Consistency propagation procedures with look-ahead heuristics, both for variable and value selection heuristics, and analyse the speedups obtained in a number of benchmark problems.

The structure of the paper is the following. In the next section we review some properties of constraint networks. In section 3 we discuss variants of Singleton Consistency, and show how to adapt them to obtain look-ahead information. In section 4 we present a number of benchmark problems and compare the results obtained when using and not using the look-ahead heuristics. In section 5 we report on related work, and finally conclude with a summary of the lessons learned and directions for further research.

2 Notation and background

A constraint network consists of a set of variables $\mathcal{X}$, a set of domains $\mathcal{D}$, and a set of constraints $\mathcal{C}$. Every variable $x \in \mathcal{X}$ has an associated domain $D(x)$ denoting its possible values. Every k-ary constraint $c \in \mathcal{C}$ is defined over a set of k variables $(x_1, \ldots, x_k)$ by the subset of the Cartesian product $D(x_1) \times \ldots \times D(x_k)$ which are consistent values. The constraint satisfaction problem (CSP) consists in finding an assignment of values to variables such that all constraints are satisfied.

A CSP is arc-consistent iff it has non-empty domains and every consistent instantiation of a variable can be extended to a consistent instantiation involving an additional variable [16]. A problem is generalized arc-consistent (GAC) iff for every value in each variable of a constraint there exist compatible values for all the other variables in the constraint.

Enforcing (generalized) arc consistency is usually not enough for solving a CSP and search must be performed. A large class of search heuristics follow

Algorithm 1 SC($\mathcal{X}$,$\mathcal{C}$) : *state*

```
do
   modified ←false
   forall x ∈ X
      modified ← sREVISE(x,X,C) ∨modified
      if D(x) = ∅
         state ←failed
         return
      endif
   endfor
while modified =true
state ←succeed
```

the first-fail/best-promise policy (FF/BP) [12], which consists of selecting the variable which more likely leads to a contradiction (FF), and then select the value that has more chances of being part of a solution (BP). For estimating first-failness, heuristics typically select the variable with smaller domain (dom), more constraints attached (deg), more constraints to instantiated variables (bdeg), or combinations (e.g. dom/deg). Best-promise is usually obtained by integrating some knowledge about the structure of the problem.

3 Look-ahead pruning algorithms

3.1 Singleton consistencies

A CSP P is singleton θ-consistent (SC), iff it has non-empty domains and for any value $a \in dom\,(x)$ of every variable $x \in \mathcal{X}$, the resulting subproblem $P|_{x=a}$ can be made θ-consistent. Most cost-effective singleton consistencies are singleton arc-consistency (SAC) [7] and singleton generalized arc-consistency (SGAC) [17].

To achieve SC in a CSP, procedure SC [7] instantiates each variable to each of its possible values in order to prune those that (after some form of propagation) lead to a domain wipe out (alg. 1). Once some (inconsistent) value is removed, then there is a chance that a value in a previously revised variable has become inconsistent, and therefore SC must check these variables again. This can happen at most nd times, where n is the number of variables, and d the size of the largest domain, hence SC time complexity is in O($n^2d^2\Theta$), Θ being the time complexity of the algorithm that achieves θ-consistency on the constraint network. Variants of this algorithm with the same pruning power but yielding distinct space-time complexity trade-offs have been proposed [1,3,4,15]. A related algorithm considers each variable only once (alg. 2), has better runtime complexity O($nd\Theta$), but achieves a weaker consistency, called restricted singleton consistency (RSC) [17].

Note that both algorithms use function sREVISE (alg. 3) which prunes the domain of a single variable by trying all of its possible instantiations.

Algorithm 2 RSC($\mathcal{X}$,$\mathcal{C}$) : *state*

```
forall x ∈ X
   sREVISE(x,X,C)
   if D(x) = ∅
      state ←failed
      return
   endif
endfor
state ←succeed
```

Algorithm 3 sREVISE(x,$\mathcal{X}$,$\mathcal{C}$) : $modified$

```
modified ←false
forall a ∈ D(x)
   try x = a
      state ←PROPAGATE_θ(X,C)
   undo x = a
   if state =failed
      D(x) ← D(x) \ a
      modified ←true
   endif
endfor
```

3.2 Pruning decisions

Another possible trade-off between run-time complexity and pruning power is to enforce singleton consistency on a subset of variables $\mathcal{S} \subset \mathcal{X}$. We identified two possible goals which condition the selection of $\mathcal{S}$: filtering and decision making. From a filtering perspective, $\mathcal{S}$ should be the smallest subset where (restricted) singleton consistency can actually prune values, and although this is not known *a priori*, approximations are possible by exploring incrementality and value support [1,4]. On the other hand, $\mathcal{S}$ may be selected for improving the decision making process, in particular of variable selection heuristics that are based on the cardinality of the current domains. In this case, the pruning resulting from enforcing singleton consistency is used as a mechanism to break ties both in the selection of variable and in the choice of the value to enumerate.

Observing the general preference for variable heuristics which select smallest domains first, we propose defining $\mathcal{S}$ as the set of variables whose domain cardinality is below a given threshold d. We denote by RSC$_d(\mathcal{X},\mathcal{C})$ and SC$_d(\mathcal{X},\mathcal{C})$, respectively, the algorithms RSC$(\mathcal{X}_{|D|\leq d},\mathcal{C})$ and SC$(\mathcal{X}_{|D|\leq d},\mathcal{C})$, where $\mathcal{X}_{|D|\leq d}$ is the subset of variables in $\mathcal{X}$ having domains with cardinality less or equal to d.

A further step in integrating singleton consistencies with search heuristics is to explore information regarding the subproblems that are generated each time a value is tested for consistency. We propose a class of look-ahead heuristics (LA) for any CSP P which reason over the size of its solution space, given by a function $\sigma(P)$, collected while enforcing singleton consistency. Although

Algorithm 4 sReviseInfo(x,$\mathcal{X}$,$\mathcal{C}$,$info$) : $modified$

```
modified ←false
forall a ∈ D(x)
   try x = a
      state ←propagate_θ(X,C)
      info[x, a] ← collectInfo(X, C)
   undo x = a
   if state =failed
      D(x) ← D(x) \ a
      modified ←true
   endif
endfor
```

Algorithm 5 search($\mathcal{X}$,$\mathcal{C}$) : $state$

```
info ← ∅
if sc(X, C) =fail
   state ←fail
   return
endif
if ∀_x : |D(x)| = 1
   state ←succeed
   return
endif
x ← selectVariable(X, info)
a ← selectValue(x, info)
state ← search(X, C ∪ (x = a)) or search(X, C ∪ (x ≠ a))
```

there is no known polynomial algorithm for computing σ (finding the number of solutions of a CSP is a #P-complete problem), there exists a number of naive as well as more sophisticated approximation functions [10,13]. We conjecture that by estimating the size of the solution space for each possible instantiation, i.e. $\sigma\left(P|_{x=a}\right)$, there is an opportunity for making more informed decisions that will exhibit both better first-failness and best-promise behaviour. Moreover, the class of approximations of σ presented below are easy to compute, do not add complexity to the cost of generating the subproblems, and only requires a slight modification of the sRevise algorithm.

The sReviseInfo algorithm (alg. 4) stores in a table (*info*) relevant information to the specific subproblem being considered in each loop iteration. In our case, *info* is an estimation of the subproblem solution space, more formally $info[x,a] = \sigma'\left(P|_{x=a}\right)$ where $\sigma' \approx \sigma$. The table is initialized before singleton consistency enforcement, computed after propagation, and handed to the selectVariable and selectValue functions as shown in algorithm 5 . There are several possible definitions for these functions associated with how they integrate the collected information. Regarding the selection of variable for a given CSP P, we identified two FF heuristics which are cheap and easy to compute:

$$var_1(P) = \arg \min_{x \in \mathcal{X}(P)} \left(\sum_{a \in D(x)} \sigma'(P|_{x=a}) \right)$$

$$var_2(P) = \arg \min_{x \in \mathcal{X}(P)} \left(\max_{a \in D(x)} \sigma'(P|_{x=a}) \right)$$

Informally, var_1 gives preference for the variable with a smaller sum of the number of solutions for every possible instantiation, while var_2 selects the variable whose instantiation with maximum number of solutions is the minimum among all variables. For the selection of value for some variable x, a possible BP heuristic is

$$val_1(P, x) = \arg \max_{a \in D(x)} (\sigma'(P|_{x=a}))$$

which simply prefers the instantiation that prunes less solutions from the remaining search space.

Functions var_1 and val_1 correspond to the minimize promise variable heuristic and maximize promise value heuristic defined in [9]. Please note that we do not claim these are the best options for the estimation of the search space or the number of solutions. We have simply adopted them for simplicity and for testing the concept (more discussion on section 6).

4 Experimental results

A theoretical analysis on the adequacy of these heuristics as FF or BP candidates is needed, but hard to accomplish. Alternatively, in this section we attempt to give some empirical evidence of the quality of these heuristics by presenting the results of using them combined with constraint propagation and backtracking search (BT) on a set of typical CSP benchmarks.

The set of heuristics selected for comparison was chosen in order to provide some insight on the adequacy of enforcing SC on a subset of variables as a good trade-off between propagation and search and on the impact of integrating LA information in the variable and value selection heuristics. As a side effect, we tried to confirm previous results on the classes of instances where SC is cost effective and on the performance of RSC regarding SC.

As a first attempt at measuring the potential of LA heuristics, a simple measure was used for estimating the number of solutions in a given subproblem:

$$\sigma' = \sum_{x \in \mathcal{X}} \log_2(D(x))$$

which informally expresses that the number of solutions is correlated to the size of the subproblem search space[1]. Although this is a very rough estimate, we

[1] We use the logarithm since the size of search space can be a very large number.

are assuming that it could nevertheless provide valuable information to compare alternatives (see section 6).

As a baseline for comparison we used the dom variable selection heuristic (see section 2) without any kind of singleton consistency enforcing. The other elements of the test set are the possible combinations of enforcing SC, RSC, SC2 or RSC2 with the dom or LA heuristics. The SC2 and RSC2 tests implement the $\text{SC}_d(\mathcal{X},\mathcal{C})$ and $\text{RSC}_d(\mathcal{X},\mathcal{C})$ strategies with $d = 2$, the threshold for which most interesting results were obtained. The LA heuristics implement the proposed functions var_1 and val_1. Each combination is thus denoted by a+b+c, where a states the type of singleton consistency enforced (or is absent if none), b specifies the variable heuristic and c the value heuristic. For example, sc+dom+min performs singleton consistency and then instantiates the variable with the smallest domain to the minimum value in its domain.

In the following experiments all times are given in seconds, and represent the time needed for finding the first solution. The column 'ratio', when present, refers to the average CPU time of the current heuristic over the baseline, which is always the CPU time of the dom heuristic. Data presented in the following charts was interpolated using a Bézier smoothing curve.

Tests regarding sections 4.1 and 4.2 were performed on a Pentium4, 3.4GHz with 1Gb RAM, while the results presented in section 4.3 were obtained on a Pentium4, 1.7GHz with 512Mb RAM.

4.1 Graph Coloring

Graph coloring consists of trying to assign n colors to m nodes of a given graph such that no pair of connected nodes have the same color. In this section we evaluate the performance of the presented heuristics in two sets of 100 instances of 10-colorable graphs, respectively with 50 and 55 nodes, generated using Joseph Culberson's k-colorable graph generator [6].

A CSP for solving the graph coloring problem was modelled by using variables to represent each node and values to define its color. Difference binary constraints were posted for every pair of connected nodes.

The average degree of a node in the graph d, i.e. the probability that each node is connected to every other node, has been used for describing the phase transition in graph coloring problems [5]. In this experiment we started by determining empirically the phase transition to be near $d = 0.6$, and then generated 100 random instances varying d uniformly in the range $[0.5 \ldots 0.7]$.

Figure 1 compares the search effort using each heuristic on the smallest graph problem, with a timeout of 300 seconds. These results clearly divide the heuristics into two sets, the set where SC and RSC was used being much better than the other on the hard instances. Since the ranking within the best set was not so clear, a second experiment on the larger and more difficult problem was performed using only these four heuristics, with a larger timeout of 900 seconds. The results of these tests are shown graphically on fig. 2, and also given in detail in table 1. This second set of experiments shows that RSC+LA is better on

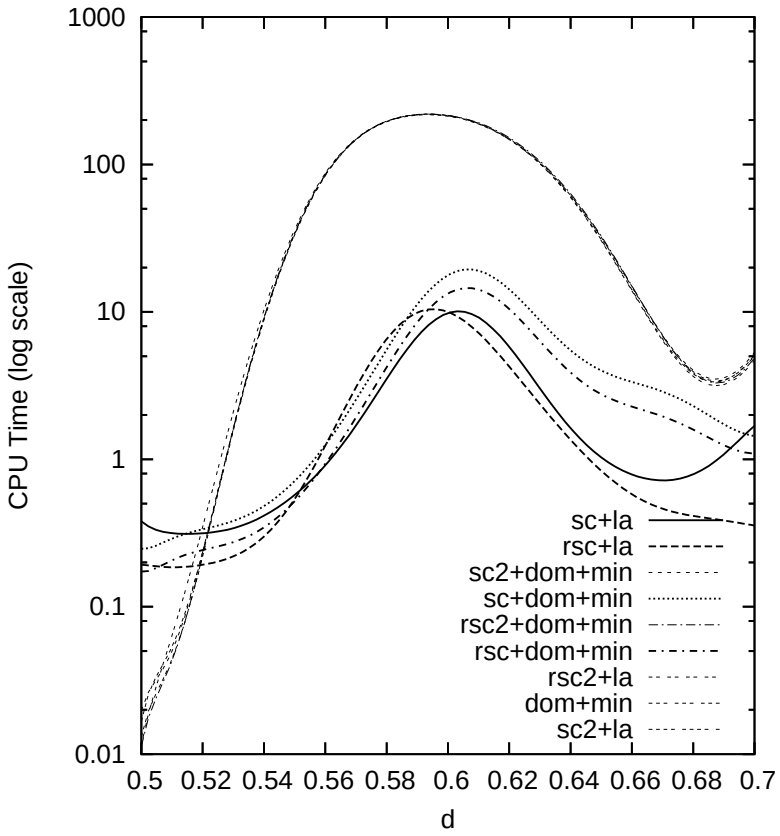

Fig. 1. CPU time spent in finding the first solution of random 10-colorable graph instances with size 50 .

heuristic	d							
	0.500 0.525	0.525 0.550	0.550 0.575	0.575 0.600	0.600 0.625	0.625 0.650	0.650 0.675	0.675 0.700
rsc+dom+min	0.71	8.46	160.59	499.27	179.93	26.60	13.59	1.00
sc+dom+min	0.93	**0.83**	183.58	624.11	184.67	46.13	21.43	0.47
rsc+la	1.18	5.03	**148.92**	**67.13**	**72.62**	**26.44**	0.73	0.72
sc+la	**0.46**	104.29	320.35	603.55	107.27	69.51	**0.36**	**0.37**

Table 1. CPU time spent in finding the first solution of random 10-colorable graph instances with size 55. Columns show averages for intervals of uniform variation of constraint tightness d.

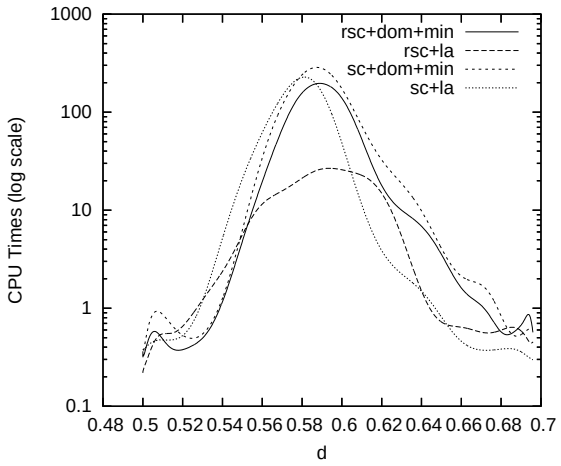

Fig. 2. CPU time spent in finding the first solution of random 10-colorable graph instances with size 55.

the most difficult instances (almost by an order of magnitude), while the others have quite similar efficiency.

4.2 Random CSPs

Randomly generated CSPs have been widely used experimentally, for instance to compare different solution algorithms. In this section we evaluate the look-ahead heuristics on several random n-ary CSPs. These problems were generated using model C [11] generalized to n-ary CSPs, that is, each instance is defined by a 5-tuple $\langle n, d, a, p_1, p_2 \rangle$, where n is the number of variables, d is the uniform size of the domains, a is the uniform constraint arity, p_1 is the density of the constraint graph, and p_2 the looseness of the constraints.

These tests evaluate the performance of the several heuristics in a set of random instances near the phase transition. For this task we used the constrainedness measure κ [10] for the case where all constraints have the same looseness and all domains have the same size:

$$\kappa = \frac{-|\mathcal{C}| \log_2 (p_2)}{n \log_2 d}$$

where $|\mathcal{C}|$ is the number of n-ary constraints.

We started by fixing n, d and a arbitrarily to 50, 5 and 3 respectively, and then computed 100 values for p_2 uniformly in the range $[0.1 \ldots 0.8]$. For each of these values, a value of p_1 was used such that $\kappa = 0.95$ (problems in the phase transition have typically $\kappa \approx 1$). The value of p_1, given by

$$p_1 = -\kappa \frac{n \log_2 d}{\log_2 p_2} \times \frac{a!\,(n-a)!}{n!}$$

is computed from the first formula and by noting that p_1 is the fraction of constraints over all possible constraints in the constraint graph, i.e.

$$p_1 = |\mathcal{C}| \frac{a!\,(n-a)!}{n!}$$

Solutions were stored as positive table constraints and GAC-Schema [2] was used for filtering. The timeout was set to 600 seconds.

Table 2 shows the results obtained. In figure 3 the performances of the most interesting heuristics are presented graphically. Besides the rsc+dom+min and

	p_2						
heuristic	0.1-0.2	0.2-0.3	0.3-0.4	0.4-0.5	0.5-0.6	0.6-0.7	0.7-0.8
dom+min	**0.06**	**0.34**	2.98	12.86	52.84	236.82	377.18
rsc2+dom+min	0.10	0.58	4.69	18.79	73.61	279.86	429.22
rsc+dom+min	0.21	1.09	5.35	25.03	97.32	319.71	471.83
sc2+dom+min	0.11	0.64	4.95	20.45	79.78	289.59	438.92
sc+dom+min	0.27	1.28	6.27	29.28	112.15	341.82	492.76
rsc2+la	**0.09**	0.45	3.47	11.85	53.64	237.19	373.80
rsc+la	0.11	**0.37**	**1.43**	**3.28**	**21.86**	**71.10**	**99.93**
sc2+la	0.10	0.50	3.78	12.92	58.60	249.03	388.77
sc+la	0.15	0.47	**1.75**	**4.04**	**26.06**	**82.60**	**115.13**

Table 2. CPU time spent in finding the first solution of random CSP instances Columns show averages for intervals of uniform variation of constraint looseness p_2.

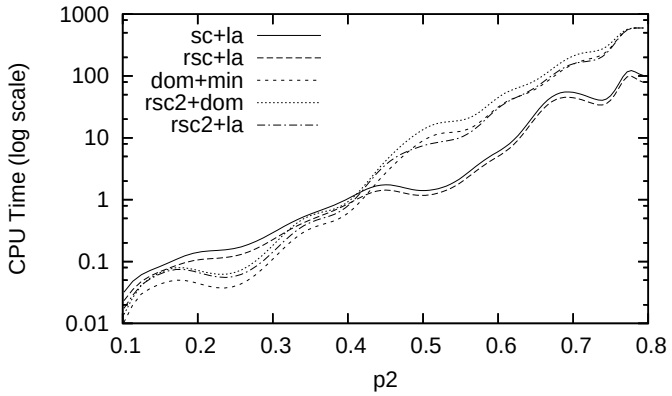

Fig. 3. CPU time spent in finding the first solution of random CSP instances.

sc+dom+min heuristics which always performed worse than the others, there

seems to be a change of ranking around $p_2 \approx 0.4$, with the dom+min dominating on the dense instances, and LA heuristics 3-4 times faster on the sparse zone. RSC+LA and SC+LA are consistently close across all instances, being RSC slightly better.

4.3 Partial Latin Squares

Latin squares is a well known benchmark which combines randomness and structure [19]. The problem consists in placing the elements $1 \dots N$ in a $N \times N$ grid, such that each element occurs exactly once on the same row or column. A partial Latin squares (or quasigroup completion) problem is a Latin squares problem with a number of preassigned cells, and the goal is to complete the puzzle.

The problem was modelled using the direct encoding, i.e. using an all-different (GAC) constraint for every row and column. The dual encoding model, as proposed in [8], was also considered but never improved over the direct model on the presented instances. The value selection heuristic used in conjunction with the dom variable selection heuristic, denoted as mc (minimum-conflicts), selects the value which occurs less in the same row and column of the variable to instantiate. This is reported to be the best known value selection heuristic for this problem [8].

We generated 200 instances of a satisfiable partial Latin squares of size 30, with 312 cells preassigned, using lsencode-v1.1 [14], a widely used random quasigroup completion problem generator. The timeout was set to 900 seconds.

Results are presented on table 3. In this problem there is a clear evidence

		#fails		time		
heuristic	#timeouts	avg	std	avg	std	ratio
dom+mc	5	18658	43583	66.7	169	1
rsc2+dom+mc	5	235	436	70.2	156.4	1.05
rsc+dom+mc	5	24	49	89.3	159.2	1.34
sc2+dom+mc	10	174	330	100.5	207.1	1.51
sc+dom+mc	6	**15**	**28**	122.8	180.3	1.84
rsc2+la	**0**	51	127	**12.2**	**19.9**	**0.18**
rsc+la	**0**	14	35	67.7	47.7	1.02
sc2+la	**0**	43	109	**15.6**	**26.4**	**0.23**
sc+la	4	**11**	**29**	104.6	134.4	1.57

Table 3. Running times and number of fails for the pls-30-312 problem. Last column shows the ratio between the average time of each heuristic over the average time of the baseline, which is the dom+mc heuristic.

of the rsc2+la and sc2+la heuristics compared to every other. Besides the fact that they are over 5 times faster than the other alternatives, they are also the most robust, as shown by their lower standard deviations as well as the absence of time out instances.

4.4 Discussion

The results obtained clarified some of the questions posed in the beginning of this section. In particular, the best performing combinations in all problems were always obtained using LA information, so this approach has clearly some potential to be explored more thoroughly.

Regarding the use of SC on a subset of variables, the results so far are not conclusive. Heuristics that restrict SC maintenance to only 2 valued variables performed badly both on the graph coloring and random problems, but clearly outperformed all others on the Latin squares problem. We think that this behaviour may be connected with the number of times these heuristics have a chance to break ties both in the selection of variable and value. The cardinality of the domains should have impact on the number of decisions having the same preference for FF heuristics, in particular the dom heuristic. The average number of values in the Latin squares problem is very low (around 3) since most variables are instantiated, so these heuristics would have more chance to make a difference here than on the other problems which have larger cardinality (5 and 10). The same argument may apply to the value selection heuristic if we note that the selection of value is more important in problems with some structure, which would again favour the Latin squares problem.

The remaining aspects of the results obtained are in accordance with the extensive analysis of singleton consistencies described in [17]. On the question of cost-effectiveness of RSC we obtained similar positive results, in fact it was slightly better than SC on all instances. Its combination with LA was the most successful, outperforming the others in the hard instances of every problem.

In the class of random problems, their work concludes that singleton consistencies are only useful in the sparse instances. Our results also confirm this. Generally, the claim that SC can be very expensive to maintain seems true in our experiments except when using combined with LA heuristics. This provides some evidence that the good behaviour of SC+LA observed relies more strongly on correct decisions rather than on the filtering achieved.

5 Related work

The work of [18] suggests improving the variable selection heuristic based on the impact each variable assignment had on past search states. The impact is defined as the ratio of search size reduction achieved when propagating the assignment. In their paper the use of a specific look-ahead procedure for measuring this impact is regarded as costly, and depreciated in favour of a method that accumulates this information across distinct search branches and/or search iterations (restarts). Their results show that the method eventually converges to a good variable ordering (the value selection heuristic is not considered).

In [13], belief updating techniques are used to estimate the likelihood of a value belonging to some solution. These likelihoods are then used to improve the value selection heuristic and as propagation: if it decreases to zero, the value is

discarded from the domain. However, the integration of this kind of propagation with common local propagation algorithms is not explored in that paper.

6 Conclusion

In this paper we presented an approach that incorporates look ahead information for directing backtracking search, and suggested that this could be largely done at no extra cost by taking advantage of the work already performed by singleton consistency enforcing algorithms. We described how such a framework could extend existing SC and RSC algorithms by requiring only minimal modifications. Additionally, a less expensive form of SC was revisited, and a new one proposed which involves revising only a subset of variables. Empirical tests with two common benchmarks and with randomly generated CSPs showed promising results on instances near the phase transition. Finally, results were analysed and matched against those previously obtained by other researchers.

There are a number of open questions and future work directions. As discussed in the previous section, tests which use singleton consistency on a subset of variables defined by its cardinality were not consistently better or worse than the others, but may be very beneficial sometimes. We think this deserves more investigation, namely testing with more structured problems, using a distinct selection criteria (other than domain cardinality), and selective performing singleton consistency less often by reusing previously computed information (in the *info* table).

The most promising direction for future work is improving the FF and BP measures. Look-ahead heuristics presented above use rather naive estimation of number of solutions for a given subproblem compared to, for example, the κ measure introduced in [10], or the probabilistic inference methods described in [13]. The κ measure, for example, takes into account the individual tightness of each constraint and the global density of the constraint graph. Their work shows strong evidence for best performance of this measure compared with standard FF heuristics, but also point out that the complexity of its computation may lead to suboptimal results in general CSP solving (the results reported are when using forward-checking). Given that we perform a stronger form of propagation and have look-ahead information available, the cost for computing κ may be worth while. We intend to investigate this in the future.

Other improvements include the use of faster singleton consistency enforcing algorithms [1,4], which should be orthogonal to the results presented here, and the use of constructive disjunction during the maintenance of SC, by pruning values from the domains of a variable that does not appear in the state of the problem for all values of another variable.

We think the results obtained so far are quite promising and justify further research along the outlined directions.

References

1. Roman Barták and Radek Erben. A new algorithm for singleton arc consistency. In Valerie Barr and Zdravko Markov, editors, *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference (FLAIRS'04), Miami Beach, Florida, USA*. AAAI Press, 2004.
2. C. Bessiére and J-C. Régin. Arc consistency for general constraint networks: preliminary results. In *Proceedings of IJCAI'97*, pages 398–404, Nagoya, Japan, 1997.
3. Christian Bessière and Romuald Debruyne. Theoretical analysis of singleton arc consistency. In *Proceedings of ECAI'04*, 2004.
4. Christian Bessière and Romuald Debruyne. Optimal and suboptimal singleton arc consistency algorithms. In *Proceedings of IJCAI'05*, 2005.
5. Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the really hard problems are. In *Proceeding of IJCAI'91*, pages 331–340, 1991.
6. Joseph Culberson. Graph coloring resources. on-line. http://web.cs.ualberta.ca/ joe/Coloring/Generators/generate.html.
7. Romuald Debruyne and Christian Bessière. Some practicable filtering techniques for the constraint satisfaction problem. In *Proceedings of IJCAI'97*, pages 412–417, 1997.
8. Dotu, del Val, and Cebrian. Redundant modeling for the quasigroup completion problem. In *ICCP: International Conference on Constraint Programming (CP), LNCS*, 2003.
9. Pieter Andreas Geelen. Dual viewpoint heuristics for binary constraint satisfaction problems. In *Proceedings of ECAI '92*, pages 31–35, New York, NY, USA, 1992. John Wiley & Sons, Inc.
10. I. P. Gent, E. MacIntyre, P. Prosser, and T. Walsh. The constrainedness of search. In *Proceedings of AAAI'96*, volume 1, pages 246–252, 1996.
11. Ian P. Gent, Ewan MacIntyre, Patrick Prosser, Barbara M. Smith, and Toby Walsh. Random constraint satisfaction: Flaws and structure. *Constraints*, 6(4):345–372, 2001.
12. R. M. Haralick and G. L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.
13. Kalev Kask, Rina Dechter, and Vibhav Gogate. New look-ahead schemes for constraint satisfaction. In *Proceeding of AMAI'04*, 2004.
14. Kautz, Ruan, Achlioptas, Gomes, Selman, and Stickel. Balance and filtering in structured satisfiable problems. In *Proceedings of IJCAI'01*, 2001.
15. Christophe Lecoutre and Stéphane Cardon. A greedy approach to establish singleton arc consistency. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *Proceedings of IJCAI-05*, pages 199–204. Professional Book Center, 2005.
16. Alan K. Mackworth and Eugene C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25:65–74, 1985.
17. Patrick Prosser, Kostas Stergiou, and Toby Walsh. Singleton consistencies. In Rina Dechter, editor, *Proceeding of CP'00*, volume 1894 of *Lecture Notes in Computer Science*, pages 353–368. Springer, 2000.
18. Philippe Refalo. Impact-based search strategies for constraint programming. In Mark Wallace, editor, *Proceedings of CP'04*, volume 3258 of *Lecture Notes in Computer Science*, pages 557–571. Springer, 2004.
19. Paul Shaw, Kostas Stergiou, and Toby Walsh. Arc consistency and quasigroup completion. In *Proceedings of ECAI'98 workshop on non-binary constraints*, March 14 1998.