ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/220481061

Activity-Based Search for Black-Box Contraint-Programming Solvers

Conference Paper · May 2011

DOI: 10.1007/978-3-642-29828-8_15 · Source: DBLP

CITATIONS
87

READS
364

2 authors:

Laurent Michel
University of Connecticut

159 PUBLICATIONS **4,191** CITATIONS

SEE PROFILE

Pascal Van Hentenryck
Georgia Institute of Technology

649 PUBLICATIONS **20,134** CITATIONS

SEE PROFILE

All content following this page was uploaded by Laurent Michel on 22 May 2014.

The user has requested enhancement of the downloaded file.

Activity-Based Search for Black-Box Constraint Programming Solvers

L. Michel[1] and P. Van Hentenryck[2]

[1] University of Connecticut, Storrs, CT 06269-2155
[2] Optimization Research Group, NICTA, Victoria Research Laboratory
The University of Melbourne, VIC 3010, Australia

Abstract. Robust search procedures are a central component in the design of black-box constraint-programming solvers. This paper proposes activity-based search which uses the activity of variables during propagation to guide the search. Activity-based search was compared experimentally to impact-based search and the WDEG heuristics but not to solution counting heuristics. Experimental results on a variety of benchmarks show that activity-based search is more robust than other heuristics and may produce significant improvements in performance.

1 Introduction

Historically, the constraint-programming (CP) community has focused on developing open, extensible optimization tools, where the modeling and the search procedure can be specialized to the problem at hand. This focus stems partially from the roots of CP in programming languages and partly from the rich modeling language typically found in CP systems. While this flexibility is appealing for experts in the field, it places significant burden on practitioners, reducing its acceptance across the wide spectrum of potential users. In recent years, the CP community devoted increasing attention to the development of black-box constraint solvers. This new focus was motivated by the success of Mixed-Integer Programming (MIP) and SAT solvers, which are typically black-box systems. As such, they allow practitioners to focus on modeling aspects.

This research is concerned with one important aspect of black-box solvers: the implementation of a robust search procedure. In recent years, various proposals have addressed this issue. Impact-based search (IBS) [12] is motivated by concepts found in MIP solvers such as strong branching and pseudo costs. Subsequent work about solution counting can be seen as an alternative to impacts [10] that exploits the structure of CP constraints. The weighted degree heuristic (WDEG) [1] inspired by [2] is a direct adaptation of the SAT heuristic VSIDS[7] to CSPs that relies on failures data to define the variable ordering.

This paper proposes Activity-Based Search (ABS), a search heuristic that recognizes the central role of constraint propagation in constraint programming systems. Its key idea is to associate with each variable a counter which measures the activity of a variable during propagation, i.e., how often (but not how much) it gets filtered by constraint propagation. This measure is updated systematically during search and initialized by a probing process. ABS has a number of advantages compared to earlier proposals. First, it does not deal explicitly with variable domains which complicates the implementation and runtime requirements of IBS. Second, it does not instrument constraints which is a significant burden in solution counting heuristics. Third, it naturally deals with global constraints, which is not the case of WDEG since all variables in a failed constraint receive the same weight contribution although only a subset of them may be relevant to the conflict. ABS was compared experimentally to IBS and WDEG on a variety of benchmarks. The results show that ABS is the most robust heuristic and can produce significant improvements in performance over IBS and WDEG, especially when the problem complexity increases.

The rest of the paper is organized as follows. Sections 2 and 3 review the IBS and WDEG heuristics. Section 4 presents ABS. Section 5 presents the experimental results and Section 6 concludes the paper.

2 Impact-Based Search

Impact-based search was motivated by the concept of pseudo-cost in MIP solvers and it associates with a branching decision $x = a$ a measure, called the *impact*, of how effectively it shrinks the search space.

Formalization Let $P = \langle X, D, C\rangle$ be a CSP defined over variables X, domains D, and constraints C. Let $D(x_i)$ denote the domain of variable $x_i \in X$ and $|D(x_i)|$ denote the size of this domain. A trivial upper-bound on the size of the search space of $\mathcal{S}(P)$ is given by the product of the domain sizes:

$$\mathcal{S}(P) = \prod_{x \in X} |D(x)|$$

At node k, the search procedure receives a CSP $P_{k-1} = \langle X, D_{k-1}, C_{k-1}\rangle$, where $C_{k-1} = C \cup \{c_0, c_1, c_2, \cdots, c_{k-1}\}$ and c_i is the constraint posted at node i. Labeling a variable x with value $a \in D_{k-1}(x) \subseteq D_(x)$ adds a constraint $x = a$ to C_{k-1} to produce, after propagation, the CSP $P_k = \langle X, D_k, C_k\rangle$.

The contraction of the search space induced by a labeling $x = a$ is defined as

$$I(x = a) = 1 - \frac{\mathcal{S}(P_k)}{\mathcal{S}(P_{k-1})}$$

$I(x = a) = 1$ when the assignment produces a failure since $\mathcal{S}(P_k) = 0$ and $I(x = a) \approx 0$ whenever $\mathcal{S}(P_k) \approx \mathcal{S}(P_{k-1})$, i.e., whenever there is almost no domain reduction. Following [12], an *estimate* of the impact of the labeling constraint $x = a$ over a set of search tree nodes $\mathcal{K}$ can be defined as the average over $\mathcal{K}$

$$\bar{I}(x = a) = \frac{\sum_{k \in \mathcal{K}} 1 - \frac{\mathcal{S}(P_k)}{\mathcal{S}(P_{k-1})}}{|\mathcal{K}|}$$

Actual implementations (e.g., [9]) rely instead on

$$\bar{I}_1(x = a) = \frac{\bar{I}_0(x = a) \cdot (\alpha - 1) + I(x = a)}{\alpha}$$

where α is a parameter of the engine and the subscripts in $\bar{I}_0$ and $\bar{I}_1$ denote the impact before and after the update. Clearly, $\alpha = 1$ yields a forgetful strategy (only the last impact is kept), $\alpha = 2$ gives a running average that decays past impacts over time, while $\alpha > 2$ favors past information over most recent observations. Both [12] and the more recent [6] adopt a pure averaging scheme.

The (approximate) impact of a variable x at node k is defined as

$$\mathcal{I}(x) = -\sum_{a \in D_k(x)} \left(1 - \bar{I}(x = a)\right) = \sum_{a \in D_k(x)} \left(\bar{I}(x = a) - 1\right)$$

$$= \left(\sum_{a \in D_k(x)} \bar{I}(x = a)\right) - |D_k(x)|$$

Namely, when all the $\bar{I}(x = a)$ are nearing 0 (no impacts) $\mathcal{I}(x)$ goes towards $-|D_k(x)|$ and when all the $\bar{I}(x = a)$ are nearing 1, $\mathcal{I}(x)$ goes to 0. Recently, Kadioglu et. al [6] suggest to exploit variance of $\bar{I}(x = a)$ to further improve the effectiveness of IBS by using the formula

$$ARF_\beta(x) = \mathcal{I}(x) + \beta \cdot \sqrt{VAR(x)}.$$[3]

To obtain suitable estimates of the assignment and variable impacts at the root node, IBS simulates all the $\sum_{x \in X} |D(x)|$ possible assignments. For large domains, domain values are partitioned in blocks. Namely, for a variable x, let $D(x) = \cup_{i=1}^{b} B_i$ with $B_i \cap B_j = \emptyset \ \forall i, j : i \neq j \in 1..b$. The impact of a value $a \in B_i$ ($i \in 1..b$) is then set to $I(x = a) = I(x \in B_i)$. With partitioning, the initialization costs drop from $|D(x)|$ propagations to b propagations (one per block). The space requirement for IBS is $\Theta(\sum_{x \in X} |D(x)|)$, since it stores the impacts of all variable/value pairs.

The Search Procedure IBS defines a variable and a value selection heuristic. IBS first selects a variable x with the largest impact, i.e., $x \in \text{argMax}_{x \in X} \mathcal{I}(x)$. It then selects a value a with the least impact, i.e., $a \in \text{argMin}_{v \in D(x)} \bar{I}(x = v)$. Neither $\text{argMax}_{x \in X} \mathcal{I}(x)$ nor $\text{argMin}_{v \in D(x)} \bar{I}(x = v)$ are guaranteed to be a singleton and, in case of ties, IBS breaks the ties uniformly at random. As any randomized search procedure, IBS can be augmented with a restart strategy. A simple restarting scheme limits the number of failures in round i to l_i and increases the limit between rounds to $l_{i+1} = \rho \cdot l_i$ where $\rho > 1$.

---

[3] Kadioglu et. al referred to [12] for the definition of $\mathcal{I}(x)$ but they use the formula $ERF(x) = 1 - \sum_{a \in D(x)} \bar{I}(x = a)$ instead in the text with $ERF(x)$ replacing $\mathcal{I}(x)$. As soon as the domain sizes start to differ, the two definitions produce different recommendations with $ERF(x)$ exhibiting a strong bias towards variables with large domains. Their experimental results seem to be based on the definition of $\mathcal{I}(x)$ from [12], the ERF formula producing poor results when used instead of $\mathcal{I}$.

3 The WDEG Heuristic

WDEG maintains, for each constraint, a counter (weight) representing the number of times the constraint has failed, i.e., the constraint removed all values in the domain of one of its variables during propagation. The weighted degree of variable x is defined as

$$\alpha_{wdeg}(x) = \sum_{c \in C} weight[c] \text{ s.t. } x \in vars(c)x \wedge |FutVars(c)| > 1$$

where $FutVars(c)$ is the set of uninstantiated variables in c. WDEG only defines a variable selection heuristic: It first selects a variable x with the smallest ratio $\frac{|D(x)|}{\alpha_{wdeg}(x)}$. All the weights are initialized to 1 and, when a constraint fails, its weight is incremented. The space overhead is $\Theta(|C|)$ for a CSP $\langle X, D, C \rangle$. Note that upon restarts the weights are not reset to 1 and restarting WDEG therefore exhibits learning as well.

4 Activity-Based Search

ABS is motivated by the key role of propagation in constraint programming solvers. Contrary to SAT solvers, CP uses sophisticated filtering algorithms to prune the search space by removing values that cannot appear in solutions. ABS exploits this filtering information and maintains, for each variable x, a measure of how often the domain of x is reduced during the search. The space requirement for this statistic is $\Theta(|X|)$. ABS can *optionally* maintain a measure of how much activity can be imputed to each assignments $x = a$ in order to drive a value-selection heuristic. If such a measure is maintained, the space requirement is proportional to the number of distinct assignments performed during the search and is bounded by $\mathcal{O}(\sum_{x \in X} |D(x)|)$. ABS relies on a decaying sum to forget the oldest statistics progressively, using an idea from VSIDS. It also initializes the activity of the variables by probing the search space.

ABS is simple to implement and does not require sophisticated constraint instrumentation. It scales to large domains without special treatment and is independent of the domain sizes when the value heuristic is not used. Also, ABS does not favor variables appearing in failed constraints, since a failure in a CP system is typically the consequence of many filtering algorithms.

Formalization Given a CSP $P = \langle X, D, C \rangle$, a CP solver applies a constraint-propagation algorithm F after a labeling decision. F produces a new domain store $D' \subseteq D$ enforcing the required level of consistency. Applying F to P identifies a subset $X' \subseteq X$ of affected variables defined by

$$\begin{aligned} \forall x \in X' \quad &: D'(x) \subset D(x); \\ \forall x \in X \setminus X' &: D'(x) = D(x). \end{aligned}$$

The *activity* of x, denoted by $A(x)$, is updated at each node k of the search tree regardless of the outcome (success or failure) by the following two rules:

$$\begin{array}{ll} \forall x \in X \text{ s.t. } |D(x)| > 1 & : A(x) = A(x) \cdot \gamma \\ \forall x \in X' & : A(x) = A(x) + 1 \end{array}$$

where X' is the subset of affected variables and γ is a decay parameter satisfying $0 \leq \gamma \leq 1$. The decay only affects free variables since otherwise it would quickly erase the activity of variables labeled early in the search.

The activity of an assignment $x = a$ at a search node k is defined as the number of affected variables in $|X'|$ when applying F on $C \cup \{x = a\}$, i.e.,

$$A_k(x = a) = |X'|.$$

As for impacts, the activity of $x = a$ over the entire tree can be estimated by an average over all the tree nodes seen so far, i.e., over the set of nodes $\mathcal{K}$. The estimation is thus defined as

$$\tilde{A}(x = a) = \frac{\sum_{k \in \mathcal{K}} A_k(x = a)}{|\mathcal{K}|}$$

Once again, it is simpler to favor a weighted sum instead

$$\tilde{A}_1(x = a) = \frac{\tilde{A}_0(x = a) \cdot (\alpha - 1) + A_k(x = a)}{\alpha}$$

where the subscripts on $\tilde{A}$ capture the estimate before and after the update.

The Search Procedure ABS defines a variable ordering and possibly a value ordering. It selects the variable x with the largest ratio $A(x)/|D(x)|$, i.e., the most active variable per domain value ($A(x)$ alone would yield a bias towards variables with large domains). Ties are broken uniformly at random. When a value heuristic is used, ABS selects a value a with the least activity, i.e., $a \in \text{argMin}_{v \in D(x)} \tilde{A}(x = v)$ as IBS would. The search procedure can be augmented with restarts. The activities can be used "as-is" to guide the search after a restart. It is also possible to reinitialize activities in various ways, but this option was not explored so far in the experimentations.

Initializing Activities ABS uses probing to initialize the activities. Consider a path π going from the root to a leaf node k in a search tree for the CSP $P = \langle X, D, C \rangle$. This path π corresponds to a sequence of labeling decisions $(x_0 = v_0, x_1 = v_1, \cdots, x_k = v_k)$ in which the j^{th} decision labels variable x_j with $v_j \in D_j(x_j)$. If $X_j \subseteq X$ is the subset of variables whose domains are filtered as a result of applying F after decision $x_j = v_j$, the activity of variable x along path π is defined as $A^\pi(x) = A^\pi_k(x)$ where

$$\begin{cases} A^\pi_0(x) = 0 \\ A^\pi_j(x) = A^\pi_{j-1}(x) + 1 \Leftrightarrow x \in X_j \ (1 \leq j \leq k) \\ A^\pi_j(x) = A^\pi_{j-1}(x) \qquad \Leftrightarrow x \notin X_j \ (1 \leq j \leq k) \end{cases}$$

$A^\pi(x) = 0$ if x was never involved in any propagation along π and $A^\pi(x) = k$ if the domain of x was filtered by each labeling decision in π. Also, $A^\pi(x) = A(x)$ when $\gamma = 1$ (no aging) and path π is followed.

Now let us now denote Π the set of all paths in some search tree of P. Each such path $\pi \in \Pi$ defines an activity $A^\pi(x)$ for each variable x. Ideally, we would want to initialize the activities of x as the average over all paths in Π, i.e.,

$$\mu_A(x) = \frac{\sum_{\pi \in \Pi} A^\pi(x)}{|\Pi|}.$$

ABS initializes the variables activities by sampling Π to obtain an estimate of the mean activity $\tilde{\mu_A}(x)$ from a sample $\tilde{\Pi} \subset \Pi$. More precisely, ABS repeatedly draws paths from Π. These paths are called *probes* and the j^{th} assignment $x_j = v_j$ in a probe p is selected uniformly at random as follows: (1) x_j is a free variable and (2) value v_j is picked from $D_j(x_j)$. During the probe execution, variable activities are updated normally but no aging is applied in order to ensure that all probes contribute equally to $\tilde{\mu_A}(x)$. Observe that some probes may terminate prematurely since a failure may be encountered; others may actually find a solution if they reach a leaf node. Moreover, if a failure is discovered at the root node, singleton arc-consistency [11] has been established and the value is removed from the domain permanently.

The number of probes is chosen to provide a good estimate of the mean activity over the paths. The probing process delivers an empirical distribution $\tilde{A}(x)$ of the activity of each variable x with mean $\tilde{\mu_A}(x)$ and standard deviation $\tilde{\sigma_A}(x)$. Since the probes are i.i.d., the distribution can be approximated by a normal distribution and the probing process is terminated when the 95% confidence interval of the t-distribution, i.e., when

$$[\tilde{\mu_A}(x) - t_{0.05,n-1} \cdot \frac{\tilde{\sigma_A}(x)}{\sqrt{n}}, \tilde{\mu_A}(x) + t_{0.05,n-1} \cdot \frac{\tilde{\sigma_A}(x)}{\sqrt{n}}]$$

is sufficiently small (e.g., within δ% of the empirical mean) for all variables x with n being the number of probes,

Observe that this process does not require a separate instrumentation. It uses the traditional activity machinery with $\gamma = 1$. In addition, the probing process does not add any space requirement: the sample mean $\tilde{\mu_A}(x)$ and the sample standard deviation $\tilde{\sigma_A}(x)$ are computed incrementally, including the activity vector A^p for each probe as it is completed. If a value heuristic is used the sampling process also maintains $A(x = a)$ for every labeling decision $x = a$ attempted during the probes.

5 Experimental Results

5.1 The Experimental Setting

The Configurations All the experiments were done on a Macbook Pro with a core i7 at 2.66Ghz running MacOS 10.6.7. IBS, WDEG, and ABS were all implemented in the COMET system [4]. Since the search algorithms are in general randomized, the empirical results are based on 50 runs and the tables report the average (μ_T) and the standard deviation σ_T of the running times in seconds. A timeout of 5 minutes was used and runs that timeout were assigned a $300s$ runtime. In the following, several variants of IBS are evaluated. IBS04 refers to the original version from [12]. IBS refers to the version found in [9] with a blending parameter $\alpha = 8$. IBS-L1 and IBS-L2 are based on the "lucky" versions ("lucky" prefers variables with large standard deviation) from [6] with $\beta = -1$ (respectively, $\beta = -2$) in the definition of $ARF_\beta(x)$. For ABS the values $\alpha = 8$, $\gamma = 0.999$ (slow aging), and $\delta = 20\%$ (the confidence interval for probing) are used throughout. Experimental results on the sensitivities of these parameters are also reported. For every heuristic, the results were obtained for three strategies, namely: no restarts (NR), fast restarting ($\rho = 1.1$) and slow restarting ($\rho = 2$). Space limitations force us to only show the best variant for IBS and WDEG but all restarting variants were evaluated. The initial failure limit is set to $3 \cdot |X|$.

Search Algorithms The search algorithms were run on the exact same models, with a single line changed to select the search procedure. In our experiments, IBS does *not* partition the domains when initializing the impacts and *always* computes the impacts exactly. Both the variable and value heuristics break ties randomly. In WDEG, no value heuristic is used: the values are tried in the sequential order of the domain. Ties in the variable selection are broken randomly. All the instances are solved using the same parameter values as explained earlier. No comparison with model-counting heuristic is provided, since these are not available in publicly available CP solvers.

Benchmarks The experimental evaluation uses five benchmarks that have been widely studied, often by different communities. The multi-knapsack and magic square problems both come from the IBS paper [12]. The progressive party has been a standard benchmark in the local search, mathematical-programming, and constraint programming communities, and captures a complex, multi-period allocation problem. The nurse rostering problem [13] originated from a math-programming paper and constraint programming was shown to be a highly effective and scalable approach. The radiation problem is taken from the 2008 MiniZinc challenge [8] and has also been heavily studied. The Costas array was used to evaluate the variance-enhanced IBS [6]. These benchmarks typically exploit many features of constraint programming systems including numerical, logical, reified, element, and global constraints.

5.2 The Core Results

Multi-Knapsack This benchmark is from [12]. The satisfaction model uses an arithmetic encoding of the binary knapsacks (not a global constraint) where the objective is replaced by a linear equality with a right-hand-side set to the known optimal value. All the constraints use traditional bound-consistency algorithms for filtering linear constraints. A second set of experiments considers

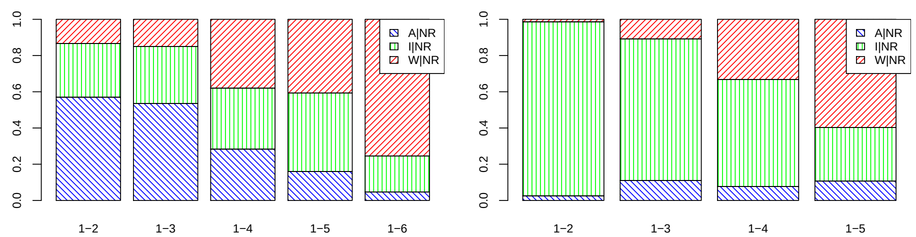


Fig. 1. Knapsack, no-restart, decision variant (left) and Optimization variant (right).

B		CSP			COP		
Bench	Model	$\mu(T)$	$\sigma(T)$	F	$\mu(T)$	$\sigma(T)$	F
1-2	Abs|NR	0.01	0.01	50	0.97	0.13	50
	Abs|R(2)	0.01	0.01	50	0.74	0.08	50
	Ibs|NR	0.01	0	50	36.61	15.19	50
	Ibs|R(2)	0.01	0	50	18.84	5.98	50
	wdeg|NR	0	0	50	0.52	0.14	50
	wdeg|R(2)	0	0	50	0.60	0.11	50
1-3	Abs|NR	0.04	0.01	50	2.03	0.27	50
	Abs|R(2)	0.04	0.01	50	1.85	0.20	50
	Ibs|NR	0.02	0.01	50	14.45	8.63	50
	Ibs|R(2)	0.03	0.01	50	14.01	10.15	50
	wdeg|NR	0.01	0.01	50	2	0.47	50
	wdeg|R(2)	0.01	0.01	50	2.55	0.72	50
1-4	Abs|NR	0.13	0.03	50	26.16	7.71	50
	Abs|R(2)	0.16	0.05	50	16.35	2.11	50
	Ibs|NR	0.15	0.07	50	200.96	41.91	50

Bench		CSP			COP		
B	Model	$\mu(T)$	$\sigma(T)$	F	$\mu(T)$	$\sigma(T)$	F
1-4	Ibs|R(2)	0.20	0.1	50	199.53	69.35	45
	wdeg|NR	0.17	0.09	50	112.95	33.57	50
	wdeg|R(2)	0.25	0.17	50	195.32	36.50	48
1-5	Abs|NR	0.78	0.26	50	53.67	13.37	50
	Abs|R(2)	0.84	0.46	50	38.68	5.26	50
	Ibs|NR	2.1	1.22	50	148.89	106.74	38
	Ibs|R(2)	2.42	1.43	50	101.53	83.29	45
	wdeg|NR	1.97	0.99	50	300.01	0	0
	wdeg|R(2)	3.98	2.12	50	300.01	0	0
1-6	Abs|NR	14.48	7.55	50			
	Abs|R(2)	19.81	12.66	50			
	Ibs|NR	54.97	29.56	50			
	Ibs04|R(2)	56.23	54.33	49			
	wdeg|NR	233.61	81.65	28			
	wdeg|R(2)	289.37	31.61	7			

Table 1. Experimental Results on Multi-Knapsack.

the optimization variant. The COP uses n global binary knapsack constraints (`binaryKnapsackAtmost` in Comet) based on the filtering algorithm in [15]. These benchmarks contain up to 50 variables.

Figure 1 is a pictorial depiction of the behavior of the three search algorithms with no restarts. The chart on the left shows the decision variant while the right chart shows the optimization variant. The stacked bar chart uses a relative scale where the height of the bar is the normalized sum of the running time of all three algorithms and the length of each segment is its normalized running time. Note that adjacent bars correspond to different totals. The left chart clearly show that, as the difficulty of the problem increases, the quality of wdeg sharply decreases and the quality of Abs significantly improves. On the harder instances, Abs is clearly superior to Ibs and vastly outperforms wdeg. The right chart was produced in the same fashion and illustrates that Ibs has the best improvement as instance size increases while Abs always finishes first.

Table 1 gives the numerical results for instances $1-2$ to $1-6$. The first column specifies the instance, while the remaining columns report the average run times, the standard deviations, and the number of runs that did not time-out. The results are given for no-restart and slow-restart strategies for all heuristics. On the decision instance $1-6$, wdeg often fails to find a solution within the time limit and, in general, takes considerable time. Abs always finds solutions and is about 5 times faster than Ibs for the no-restart strategy which is most effective on the decision variant. On the optimization variant, WDEG cannot solve instance $1-5$ in any of the 50 runs and IBS does not always find a solution. ABS, in contrast, finds a solution in all 50 runs well within the time limit. The best performers on the largest instance among the 4 variants of IBS are IBS04 ([12]) when restarting slowly and the α-weighted IBS ([9]) when not restarting. The COP variant for 1-6 is not reported as none of the algorithms *proved* optimality in the allotted time. Note that, with $R = 2$, ABS *finds* the optimum within the time budget. While IBS-L1 and IBS-L2 did better than IBS04, neither overtook the α-weighted version of IBS. In all cases, ABS is the strongest performer in this group.

In summary, on this benchmark, WDEG is vastly outperformed by IBS and ABS as soon as the instances are not easy. ABS is clearly the most robust heuristic (it always finishes within the time limit) and produces significant improvements in performance on the most difficult instances, both in the decision and optimization variants.

Magic Square This benchmark is also from [12] and the model is based on a direct algebraic encoding with $2 \cdot n$ linear equations for the rows and columns (the square side is n), 2 linear equations for the diagonals, one `alldifferent` constraint (not enforcing domain consistency) for the entire square, $2 \cdot n$ binary inequalities to order the elements in the diagonals, and two binary inequalities to order the top-left corner against the bottom-left and top-right corners. Table 2 report results for squares of size 7 to size 10. The F column in Table 2 reports the number of successful runs (no timeout).

On magic squares, WDEG is completely dominated by IBS and ABS: It has poor performance and is not robust even on the simpler instances. The best performance for IBS and ABS is obtained using a fast restart, in which case ABS and IBS are virtually indistinguishable (We report the best IBS only, but all variants are really close). IBS is more effective than ABS with slow or no restarts.

Progressive Party The progressive party problem [14] is a constraint satisfaction problem featuring a mix of global constraint and has been used frequently for benchmarking CP, LS, and MIP solvers. The instance considered here is the $2-8$ instance with 29 guests, 8 periods and 13 hosts, i.e., 232 variables with domains of size 13. The goal is to find a schedule for a social event taking place over k time periods subject to constraints on the sizes of the venues (the boats), sizes of the group, and social constraints (two groups cannot meet more than once and one group cannot go back to the same boat more than once). The model relies on multiple global alldifferent, multi-knapsacks and arithmetic constraints with reifications. This model breaks the search in k phases (one per period) and uses the black-box heuristic within each period.

The results are given in Table 3 and include all versions of IBS. ABS is the overall best performer on this benchmark with the most successes within the time limit, the smaller standard deviation and the best running times. IBS04 manages a tiny advantage with restarting but exhibits a larger deviation in those cases. The "lucky" version do not overtake the α-weighted version. ABS is also clearly

B	Model	μ_C	μ_T	σ_T	F
7	Abs|NR	8218.06	0.53	1.54	50
	Abs|R(1.1)	2094.56	0.212	0.12	50
	Abs|R(2)	2380.06	0.24	0.11	50
	Ibs|NR	1030.8	0.09	0.04	50
	Ibs|R(1.1)	1172.88	0.17	0.08	50
	Ibs|R(2)	961.78	0.11	0.05	50
	wdeg|NR	3294520	105.48	138.24	34
	wdeg|R(1.1)	4144754.2	146.25	142.82	30
	wdeg|R(2)	218408.26	8.03	42.77	49
8	Abs|NR	154783.76	7.52	42.36	49
	Abs|R(1.1)	5084.18	0.48	0.24	50
	Abs|R(2)	5941.92	0.48	0.37	50
	Ibs|NR	1889.4	0.21	0.16	50
	Ibs|R(1.1)	2694.34	0.50	0.24	50
	Ibs|R(2)	2524.08	0.31	0.22	50
	wdeg|NR	2030330.7	79.24	127.69	38
	wdeg|R(1.1)	644467.4	28.77	79.38	47
	wdeg|R(2)	339115.4	14.96	59.24	48

B	Model	μ_C	μ_T	σ_T	F
9	Abs|NR	624461.66	37.21	95.16	45
	Abs|R(1.1)	12273.72	0.96	0.66	50
	Abs|R(2)	17552.9	1.15	1.08	50
	Ibs|NR	630145.78	34.19	91.62	45
	Ibs|R(1.1)	7239.14	1.37	0.73	50
	Ibs|R(2)	7622.44	0.87	1.12	50
	wdeg|NR	5178690.7	243.38	113.08	11
	wdeg|R(1.1)	3588191.4	201.01	126.23	22
	wdeg|R(2)	1930318.7	96.67	131.75	36
10	Abs|NR	856210.12	55.01	111.94	42
	Abs|R(1.1)	32404.9	2.59	2.18	50
	Abs|R(2)	43621.08	3.24	5.04	50
	Ibs04|NR	509230.94	27.90	83.36	46
	Ibs04|R(1.1)	17253.80	2.40	1.18	50
	wdeg|NR	4508166.9	245.92	112.18	10
	wdeg|R(1.1)	-	-	-	0
	wdeg|R(2)	1825065.5	99.70	125.56	34

Table 2. Experimental Results on Magic Squares.

Row Labels	$\mu(C)$	$\mu(T)$	$\sigma(T)$	F
Abs|NR	153848.80	46.49	90.24	45
Abs|R(1.1)	2338.18	4.91	0.87	50
Abs|R(2)	4324.88	5.47	2.10	50
Ibs04|NR	971906.60	93.50	130.17	37
Ibs04|R(1.1)	11150.64	3.56	1.21	50
Ibs04|R(2)	18376.48	4.07	2.87	50

Row Labels	$\mu(C)$	$\mu(T)$	$\sigma(T)$	F
Ibs|NR	932028.14	100.74	129.24	36
Ibs|R(1.1)	16873.54	4.91	2.03	50
Ibs|R(2)	28242.14	5.74	4.85	50
wdeg|NR	405027.32	93.91	128.18	37
wdeg|R(1.1)	14424.60	3.49	2.79	50
wdeg|R(2)	19594.12	4.00	4.50	50

Table 3. Experimental Results on the Progressive Party $2 - 8$.

superior to wdeg when no restarts are used but is slightly slower than wdeg when slow or fast restarts are used.

Nurse Rostering This benchmark is taken from [13] and is a rostering problem assigning nurses to infants in an hospital ward, while balancing the workload. The multi-zone model can be found in Listing 1.2 in [13]. The custom search procedure is removed and replaced by a call to one of the generic searches (Ibs,Abs,wdeg). Table 4 reports the performance results for the three heuristics and 3 restarting strategies on the one-zone instances (z1-z5,z8). Note that the custom procedure in [13] relies on a dynamic-symmetry breaking on values and sophisticated variable/value ordering. Results for wdeg beyond `z5` are not reported as it times out systematically. As before, column F reports the number of runs that finish (out of 50), C reports the number of choice points and the T columns reports the mean and standard deviation of the running time.

wdeg exhibits extremely poor performance and robustness on this benchmark. Abs is clearly the most robust procedure as it solves all instances in all its runs for all the restarting strategies. It is also significantly faster than Ibs on z4 and z8 but slower on z5. The fastest Ibs variant changes depending on the restarting strategy. When not restarting, the "lucky" variant takes the top honor with 165 seconds on average and 50 runs. Without restarts, Abs terminates the same task in 3.5 seconds on average with the same perfect success score.

B	Model	$\mu(C)$	$\mu(T)$	$\sigma(T)$	F	B	Model	$\mu(C)$	$\mu(T)$	$\sigma(T)$	F
z1	Abs|NR	282.12	0.02	0.00	50	z3	wdeg|NR	4679035.24	300.00	0.00	2
	Abs|R(1.1)	235.52	0.02	0.01	50		wdeg|R(1.1)	5517976.00	300.00	0.00	0
	Abs|R(2)	267.58	0.02	0.01	50		wdeg|R(2)	4812533.43	300.00	0.00	2
	Ibs|NR	1113.26	0.07	0.01	50	z4	Abs|NR	30221.04	1.41	0.09	50
	Ibs|R(1.1)	1028.38	0.08	0.01	50		Abs|R(1.1)	257205.36	11.60	0.21	50
	Ibs|R(2)	820.52	0.07	0.01	50		Abs|R(2)	54855.60	2.53	0.08	50
	wdeg|NR	45043.22	1.77	0.08	50		Ibs|NR	2782779.16	106.84	29.95	50
	wdeg|R(1.1)	63783.44	2.46	0.17	50		Ibs|R(1.1)	7388602.08	300.00	0.00	2
	wdeg|R(2)	47162.36	1.87	0.08	50		Ibs|R(2)	5880894.18	237.20	40.04	48
z2	Abs|NR	45223.02	2.42	0.65	50		wdeg|NR	6386541.00	300.00	0.00	0
	Abs|R(1.1)	372174.98	19.49	9.03	50		wdeg|R(1.1)	5707406.00	300.00	0.00	0
	Abs|R(2)	98057.72	5.03	2.53	50		wdeg|R(2)	5000897.00	300.00	0.00	0
	Ibs|NR	82182.32	3.84	0.91	50	z5	Abs|NR	344187.52	17.89	3.91	50
	Ibs|R(1.1)	656035.56	24.86	7.60	50		Abs|R(1.1)	3899344.36	185.81	38.09	50
	Ibs|R(2)	177432.42	6.78	1.96	50		Abs|R(2)	902142.38	43.40	12.82	50
	wdeg|NR	6361685.84	300.00	0.00	1		Ibs|NR	114692.60	6.26	4.16	50
	wdeg|R(1.1)	5372380.94	300.00	0.00	3		Ibs|R(1.1)	423636.56	24.30	6.80	50
	wdeg|R(2)	4944998.26	300.00	0.00	1		Ibs|R(2)	176624.20	9.79	5.59	50
z3	Abs|NR	326902.20	23.32	10.88	50	z8	Abs|NR	59314.68	3.52	0.18	50
	Abs|R(1.1)	1944533.10	139.55	81.15	50		Abs|R(1.1)	599777.70	36.04	3.70	50
	Abs|R(2)	488344.88	35.26	25.40	50		Abs|R(2)	119224.04	7.00	0.53	50
	Ibs|NR	214032.16	14.96	4.45	50		Ibs|R(1.1)	8501205.52	296.51	15.42	5
	Ibs|R(1.1)	893297.88	62.27	12.23	50		Ibs|R(2)	3918758.98	146.10	44.69	47
	Ibs|R(2)	287935.30	19.62	7.01	50		Ibs-L2|NR	2549952.84	165.46	53.22	50

Table 4. Experimental Results on Nurse Rostering.

B	$\sum x \times \|D(x)\|$	Total
6	1x144 + 10x37 + 296x37	11466
7	1x178 + 10x37 + 333x37	12869
8	1x149 + 10x37 + 333x37	12840
9	1x175 + 10x37 + 333x37	12866
10	1x233 + 10x50 + 441x50	22783

Table 5. Description of the Radiation Instances.

Radiation This last benchmark is a constrained optimization problem for *radiation therapy* taken from the 2008 MiniZinc challenge [8]. The objective is to find a setup of a radiation therapy machine to deliver a desired radiation intensity to a tumor. The problem uses algebraic constraint and the formulation can be found in the mini-zinc repository [5][4]. The search procedure must deal with all the variables at once, i.e., the search was not manually broken down in phases as is done in the MiniZinc model. In 2008, several solvers were unable to solve most instances in a reasonable amount of time as seen in [5], which indicates the difficulty of the instances. The instance sizes are specified in Table 5. A row gives a term for each array in the problem with its size and the size of the domains while the last column is the corresponding value. Instance 9 has one variable with domain size 175 and 10+333 variables of size 37.

Table 6 reports the results for 5 instances. Abs clearly dominates Ibs on all instances and Ibs cannot solve the largest instance within the time limit for any restarting strategy. wdeg performs well in general on this benchmark. It is faster than Abs on the largest instance with restarts, but slower without. Both wdeg and Abs are effective on this problem and clearly superior to Ibs.

Costas Array This benchmark was used in [6] for the evaluation of the variance-enhanced Ibs and was therefore included here as well. Lack of space prevents us

[4] In this model, the time that the beam is on is a variable and must be optimized too.

B	Model	$\mu(C)$	$\mu(T)$	$\sigma(T)$	F
6	Abs|NR	14934.94	1.99	0.65	50
	Abs|R(1.1)	10653.36	1.49	0.39	50
	Abs|R(2)	10768.98	1.50	0.44	50
	Ibs|NR	65418.78	6.89	0.72	50
	Ibs|R(1.1)	86200.18	8.60	0.98	50
	Ibs|R(2)	67003.40	7.07	0.70	50
	wdeg|NR	23279.70	1.77	0.41	50
	wdeg|R(1.1)	3798.00	0.30	0.12	50
	wdeg|R(2)	2918.68	0.23	0.08	50
7	Abs|NR	17434.30	2.73	1.84	50
	Abs|R(1.1)	8481.62	1.53	0.35	50
	Abs|R(2)	9229.80	1.62	0.51	50
	Ibs|NR	90055.32	10.42	0.44	50
	Ibs|R(1.1)	161022.24	15.93	6.43	50
	Ibs|R(2)	98742.94	11.13	1.73	50
	wdeg|NR	7868.16	0.65	0.24	50
	wdeg|R(1.1)	2762.26	0.24	0.10	50
	wdeg|R(2)	2824.00	0.24	0.12	50
8	Abs|NR	33916.58	4.31	1.04	50
	Abs|R(1.1)	48638.90	6.01	0.89	50
	Abs|R(2)	18294.96	2.46	0.52	50
	Ibs|NR	84329.16	8.98	1.08	50
	Ibs|R(1.1)	187346.80	16.94	4.97	50

B	Model	$\mu(C)$	$\mu(T)$	$\sigma(T)$	F
8	Ibs|R(2)	88117.48	9.36	1.34	50
	wdeg|NR	38591.42	2.90	0.58	50
	wdeg|R(1.1)	20396.80	1.72	0.39	50
	wdeg|R(2)	6907.14	0.55	0.12	50
9	Abs|NR	40339.62	5.79	3.36	50
	Abs|R(1.1)	20599.88	3.21	0.35	50
	Abs|R(2)	14101.00	2.28	0.51	50
	Ibs|NR	85205.62	9.70	0.61	50
	Ibs|R(1.1)	141311.76	14.40	3.03	50
	Ibs|R(2)	92431.06	10.34	0.60	50
	wdeg|NR	90489.62	7.33	1.35	50
	wdeg|R(1.1)	48641.80	4.49	1.73	50
	wdeg|R(2)	12806.06	1.20	0.58	50
10	Abs|NR	210181.18	34.56	17.00	50
	Abs|R(1.1)	102777.38	17.19	3.53	50
	Abs|R(2)	50346.82	9.10	1.65	50
	Ibs|NR	2551543.8	300.01	0.00	0
	Ibs|R(1.1)	2504564.1	300.01	0.00	0
	Ibs|R(2)	2525199.8	300.01	0.00	0
	wdeg|NR	629073.46	60.09	39.47	49
	wdeg|R(1.1)	232572.16	27.88	2.28	50
	wdeg|R(2)	47175.04	5.60	1.30	50

Table 6. Experimental Results on Radiation Benchmarks.

from including a detailed table and we briefly summarize the results for size 15. Without restarts, Abs is about 3 times faster than the best Ibs variant (10.7s vs. 30.9s) while Abs is only slightly ahead of wdeg which terminates in 17s on average. With restarts, the three heuristics improve with wdeg closing the gap on Abs and even taking the lead with slow restarts. Ibs also improves, but it remains about 3 times slower than Abs regardless of the restarting speed.

Summary On this collection of benchmarks, Abs is clearly the most robust and effective heuristic. It is robust across all benchmarks and restarting strategies and is, in general, the fastest. wdeg has significant robustness and performance issues on the multi-knapsack, magic square, and nurse rostering benchmarks. Ibs has some robustness issues on radiation, some rostering instances, and the optimization variant of the large knapsack problems. It is in general significantly less efficient than Abs on the knapsack, rostering, and radiation benchmarks.

5.3 Sensitivity Analysis

Criticality of the Variable Ordering Table 7 reports the performance of activity-based search when no value ordering is used on the radiation benchmarks. The value heuristic simply tries the value in increasing order as in wdeg. The results indicate that the value selection heuristic of Abs does not play a critical role and is only marginally faster/more robust on the largest instances.

Sensitivity to the Sample Size Figure 2 illustrates graphically the sensitivy of Abs to the confidence interval parameter δ used to control the number of probes in the initialization process. The statistics are based on 50 runs of the non-restarting strategy. The boxplots show the four main quartiles for the running

B	Method	$\mu(C)$	$\mu(T)$	$\sigma(T)$	S
6	ABS|NR	11224.80	1.48	0.58	50
	ABS|R(1.1)	18803.18	2.30	0.86	50
	ABS|R(2)	12248.46	1.57	0.43	50
7	ABS|NR	7147.90	1.27	0.39	50
	ABS|R(1.1)	12161.34	1.92	0.68	50
	ABS|R(2)	10926.12	1.74	0.54	50
8	ABS|NR	27702.00	3.53	0.78	50
	ABS|R(1.1)	63755.24	7.80	2.27	50
8	ABS|R(2)	16708.46	2.23	0.47	50
9	ABS|NR	36534.92	5.06	1.18	50
	ABS|R(1.1)	46948.84	6.76	1.99	50
	ABS|R(2)	23600.68	3.46	1.02	50
10	ABS|NR	213094.82	33.70	9.23	50
	ABS|R(1.1)	239145.34	40.75	7.55	50
	ABS|R(2)	87626.36	14.87	4.14	50

Table 7. The Influence of the Value Ordering On `radiation`.

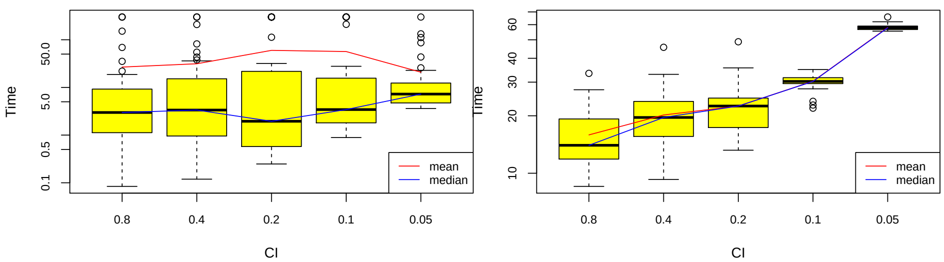


Fig. 2. Sensitivity to the Sample Size as Specified by δ.

time (in seconds) of ABS with δ ranging from 0.8 down to 0.05. The blue line connects the medians whereas the red line connects the means. The circles beyond the extreme quartiles are outliers. The left boxplot shows results on `msq-10` while the right one shows results on the optimization version of `knap1-4`.

The results show that, as the number of probes increases (i.e., δ becomes smaller), the robustness of the search heuristic improves and the median and the mean tend to converge. This is especially true on `knap1-4`, while `msq-10` still exhibits some variance when $\delta = 0.05$. Also, the mean decreases with more probes on `msq-10`, while it increases on `knap1-4` as the probe time gets larger. The value $\delta = 0.2$ used throughout seem to be a reasonable compromise.

Sensitivity to γ (Aging) Figure 3 illustrates the sensitivity to aging. The two boxplots are showing the distribution of running times in seconds for 50 runs of `msq-10` (left) and `knap1-4` (right). What is not immediately visible on the figure is that the number of timeouts for `msq-10` increases from 0 for $\gamma = 0.999$ to 9 for $\gamma = 0.5$. Overall, the results seem to indicate that slow aging is desirable.

5.4 Some Behavioral Observations

Figure 5 depicts striking behavior of ABS and IBS on `radiation #9` under all three restarting strategies. The x axis is the running time in a logarithmic scale and the y axis is the objective value each time a new upper bound is found. The three 'bottom' curves depict the performance of ABS, while the three 'top' curves correspond to IBS. ABS quickly dives to the optimal solution and spends the remaining time proving optimality. Without restarts, ABS hits the optimum within 3 seconds. With restarts, it finds the optimal within one second and the proof of optimality is faster too. IBS slowly reaches the optimal solution but

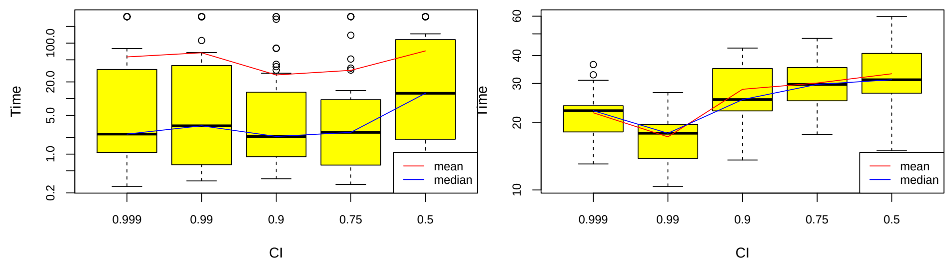
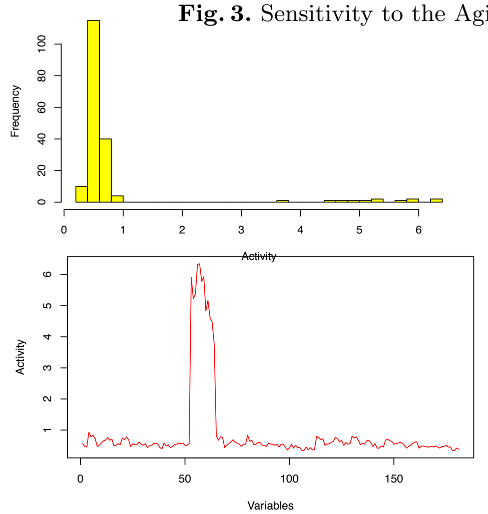


Fig. 3. Sensitivity to the Aging Process as Specified by γ.



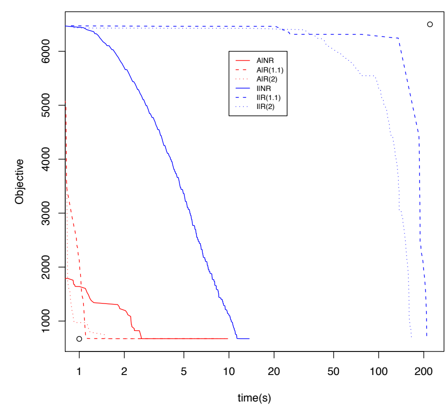


Fig. 4. Activity Freq. & Distr. on `rad-9`.

Fig. 5. The `rad-9` Objective over Time.

proves optimality quickly. Restarts have a negative effect on IBS. We conjecture that the reduction of large domains may not be such a good indicator of progress.

Figure 4 provide interesting data about activities on `radiation #9`. It gives the frequencies (an histogram of activity with buckets of size 0.2) of activity levels at the root, and plots the activity levels for all variables. (Only those not fixed by singleton arc-consistency). The figures highlight that the probing process isolates a small subset of the variables with very high activity levels. It is tempting to conjecture that this benchmark has backdoors [16] or good cycle-cutsets [3] that ABS was able to discover, but more experiments are needed to confirm or disprove this conjecture.

6 Conclusion

Robust search procedures is a central component in the design of black-box constraint programming solvers. This paper proposed activity-based search, the idea of using the activity of variables during propagation to guide the search. A variable activity is incremented every time the propagation step filters its domain and is aged. A sampling process initializes the variable activities prior to search. ABS was compared experimentally to IBS and WDEG on a variety of benchmarks. The experimental results have shown that ABS was significantly more robust than both IBS and WDEG on these benchmarks and often produces significant performance improvements.

References

1. F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In R. L. de Mántaras and L. Saitta, editors, *ECAI*, pages 146–150. IOS Press, 2004.
2. L. Brisoux, É. Grégoire, and L. Sais. Improving backtrack search for SAT by means of redundancy. In *Foundations of Intelligent Systems*, pages 301–309. Springer Berlin / Heidelberg, 1999.
3. R. Dechter and J. Pearl. The cycle-cutset method for improving search performance in ai applications. In *Proceedings of* 3^{rd} *IEEE conference on AI Applications*, Orlando, FL, 1987.
4. I. Dynadec. Comet v2.1 user manual. Technical report, Providence, RI, 2009.
5. G12. `http://www.g12.cs.mu.oz.au/minizinc/`, 2008.
6. S. Kadioglu, E. O'Mahony, P. Refalo, and M. Sellmann. Incorporating variance in impact-based search. In J. H.-M. Lee, editor, *CP*, volume 6876 of *Lecture Notes in Computer Science*, pages 470–477. Springer, 2011.
7. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: engineering an efficient sat solver. In *Proceedings of the 38th annual Design Automation Conference*, DAC '01, pages 530–535, New York, NY, USA, 2001. ACM.
8. N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack. Minizinc: Towards a standard cp modelling language. In *In: Proc. of 13th International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007.
9. L. Perron. `http://code.google.com/p/or-tools/`, 2010.
10. G. Pesant. Counting and estimating lattice points: Special polytopes for branching heuristics in constraint programming. *Optima Newsletter*, 81:9–14, 2009.
11. P. Prosser, K. Stergiou, and T. Walsh. Singleton consistencies. In *CP'02*, pages 353–368, London, UK, 2000. Springer-Verlag.
12. P. Refalo. Impact-based search strategies for constraint programming. In M. Wallace, editor, *CP*, volume 3258 of *Lecture Notes in Computer Science*, pages 557–571. Springer, 2004.
13. P. Schaus, P. Van Hentenryck, and J.-C. Rgin. Scalable load balancing in nurse to patient assignment problems. In W.-J. van Hoeve and J. Hooker, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 5547 of *Lecture Notes in Computer Science*, pages 248–262. Springer Berlin / Heidelberg, 2009.
14. B. Smith, S. Brailsford, P. Hubbard, and H. Williams. The Progressive Party Problem: Integer Linear Programming and Constraint Programming Compared. *Constraints*, 1:119–138, 1996.
15. M. A. Trick. A dynamic programming approach for consistency and propagation for knapsack constraints. In *Annals of Operations Research*, pages 113–124, 2001.
16. R. Williams, C. P. Gomes, and B. Selman. Backdoors to typical case complexity. In *Proceedings of the 18th international joint conference on Artificial intelligence*, pages 1173–1178, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc.

View publication stats