

The Impact of Search Heuristics on Heavy-Tailed Behaviour

Tudor Hulubei and Barry O’Sullivan

Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
tudor@hulubei.net, b.osullivan@cs.ucc.ie

Abstract. The heavy-tailed phenomenon that characterises the runtime distributions of backtrack search procedures has received considerable attention over the past few years. Some have conjectured that heavy-tailed behaviour is largely due to the characteristics of the algorithm used. Others have conjectured that problem structure is a significant contributor. In this paper we attempt to explore the former hypothesis, namely we study how variable and value ordering heuristics impact the heavy-tailedness of runtime distributions of backtrack search procedures. We demonstrate that heavy-tailed behaviour can be eliminated from particular classes of random problems by carefully selecting the search heuristics, even when using chronological backtrack search. We also show that combinations of good search heuristics can eliminate heavy tails from quasigroups with holes of order 10 and 20, and give some insights into why this is the case. These results motivate a more detailed analysis of the effects that variable and value orderings can have on heavy-tailedness. We show how combinations of variable and value ordering heuristics can result in a runtime distribution being *inherently heavy-tailed*. Specifically, we show that even if we were to use an oracle to refute insoluble subtrees optimally, for some combinations of heuristics we would still observe heavy-tailed behaviour. Finally, we study the distributions of refutation sizes found using different combinations of heuristics and gain some further insights into what characteristics tend to give rise to heavy-tailed behaviour.

1 Introduction

The Italian-born Swiss economist Vilfredo Pareto first introduced the theory of non-standard probability distributions in 1897 in the context of income distribution. These distributions have been used to model many real-world phenomena, from weather forecasting to stock market analysis. More recently, they have been used to model the cost of combinatorial search methods. Exceptionally hard instances have been observed amongst certain classes of constraint satisfaction problems, such as graph colouring [17], SAT [10], random problems [2, 13, 25, 26], and quasigroup completion problems [15]. In studying this phenomenon, researchers have used a wide range of systematic search algorithms such as chronological backtracking, forward-checking, Davis-Putnam and the *Maintaining Arc Consistency* algorithm (MAC) [24]. It is widely believed that the more sophisticated the search algorithm, the less likely it is that the exceptionally hard problem instances will be observed [8, 13].

Instances that are exceptionally hard occur in the under-constrained area and are often harder than those in the critically constrained region [26]. For a proper understanding of search behaviour one must study the runtime distributions [13] associated with either repeatedly solving a single instance with a randomised algorithm, or with solving a large ensemble of instances of some class of problems. Some runtime distributions exhibit an extremely large variance that can be described by heavy-tailed distributions whose tails have power-law decay. Gomes et al. [14] provided an overview of the heavy-tailed behaviour previously observed near the phase transition in NP-complete problems and introduced a rapid randomised restarts strategy aimed at avoiding the long tails. More recently, Gomes et al. [12] studied the transition between heavy-tailed and non-heavy-tailed behaviour in runtime distributions for random problems and have characterised when the phenomenon occurs and when it does not.

The motivation behind the work we present in this paper comes from a number of interesting observations we made while reproducing the random problem experiments presented by Gomes et al., particularly while studying heavy-tailed behaviour in the context of MAC. We observed that once we enhanced MAC with any of the well-known standard variable ordering heuristics, such as min-domain [16], min-dom/ddeg¹ [3], brelaz [5] and min-dom/wdeg [4, 23], heavy tails cannot be observed even for problems with 100 variables, for any density and tightness setting. Moreover, for the random problems used by Gomes et al. in their experiments with chronological backtrack search, we no longer observed heavy tails when we used min-dom/wdeg.

It has been conjectured that heavy-tailed behaviour is largely due to the characteristics of the algorithm used to solve the problems, in particular that the search algorithm makes a mistake that results in a significant amount of work being required to recover from it [15]. More recent work has formally shown that heavy-tailed behaviour can arise in backtrack search when the search tree is highly irregular and imbalanced [6]. We instead focus empirically on how heuristics can affect heavy-tailed behaviour, but make no attempt to study the structure of the resulting search tree.

In this paper we show how variable and value ordering heuristics impact the heavy-tailed phenomenon one observes in the runtime distributions of backtrack search procedures. Our analysis focuses on the MAC algorithm while solving a large ensemble of satisfiable instances of the quasigroups with holes (QWH) problem, encoded as a binary CSP. We combine different variable and value ordering heuristics with MAC to obtain a suite of algorithms that we can study. Our approach is based on analysing the refutations of insoluble (sub)trees encountered by MAC as it finds the first solution [18].

We present the following observations and results:

1. We observe that heavy-tailed behaviour associated with the runtime distribution of MAC on QWH of order 10 (QWH-10) can be eliminated by carefully selecting the search heuristics. We also show that this remains the case, although the tail becomes fatter [11], when the problem size is increased considerably to order 20.
2. We perform a more detailed analysis of the effects that variable and value orderings have on heavy-tailedness. We show how combinations of variable and value ordering heuristics can result in a problem being *inherently heavy-tailed*. Specifically,

¹ In this paper we abbreviate dynamic-degree as ‘ddeg’ and weighted-degree as ‘wdeg’.

we show that even if we were able to use an oracle to refute insoluble subtrees optimally, for some combinations of heuristics we would still observe heavy-tailed behaviour.

3. Finally, we study the distribution of refutations found using different combinations of heuristics and gain some further insights into what characteristics tend to give rise to heavy-tailed behaviour. Such a detailed analysis is the first of its kind to be reported in the literature.

The remainder of this paper is structured as follows. In Section 2 we present the motivation for this work and summarise our results. In Section 3 we present some formal preliminaries and describe the principle that underlies the analysis we have employed, i.e. computing optimal refutation trees for insoluble subproblems. We present a summary of the algorithm used to find optimal refutations and a caching scheme used to improve its time efficiency in Section 4. A detailed summary of our experiments is presented in Section 5. A number of concluding remarks are made in Section 6.

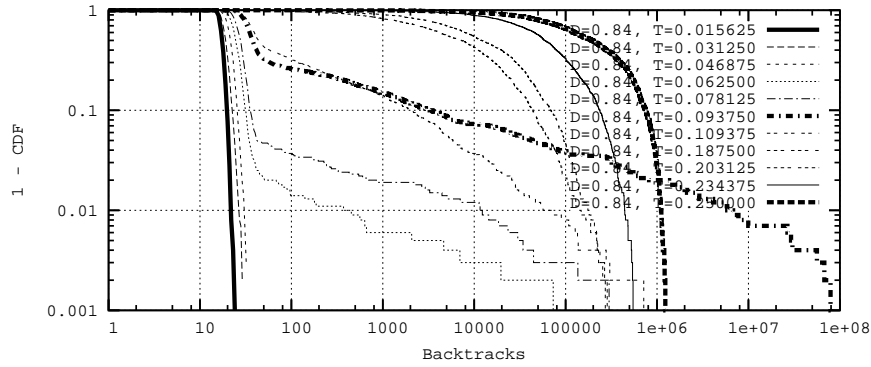
2 Motivation and Summary

Two important questions one can attempt to address when studying heavy-tailed behaviour are *when* does the phenomenon occur, and *why*. Gomes et al. [12] have characterised *when* the phenomenon occurs, in this paper we focus on studying the reasons *why*.

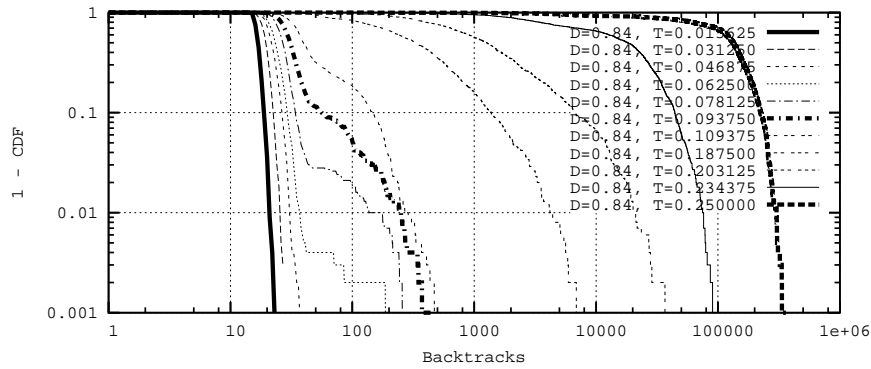
Our work is motivated by the observation that even when using chronological backtracking, for certain classes of problems heavy tails can be eliminated by using carefully chosen search heuristics. Runtime distributions are sometimes characterised by long tails, or *heavy tails*, which are generally modelled using the expression $1 - F(x) = P\{X > x\} \sim Cx^{-\alpha}, x > 0$, where $F(x)$ is the cumulative distribution function (CDF) of a probability distribution function $f(x)$ (PDF), and C and α are constants with $\alpha \in (0, 2)$ and $C > 0$. A near-straight line in a log-log plot over several orders of magnitude for $1 - F(x)$, with a slope equal to $-\alpha$, is a clear sign of heavy-tailed behaviour. From an empirical perspective, we visually check for straightness in the log-log plot and, if that is the case, we formally estimate the index of stability using an adaptation of the Hill estimator designed for truncated data [11, 15].

In Figure 1(a) we present results similar to those presented by Gomes et al. [12]. In this figure we use a chronological backtrack search procedure that uses both random variable and value orderings, solving problems with different levels of constrainedness (Model B [9]; instances with 17 variables, 8 values, density 0.84, and tightness between 0.015 and 0.25, the point where the phase transition occurs). This figure shows that heavy-tailed behaviour ($\alpha = 0.481$) can be observed in problems that are in the easy region, far from the phase transition, but are not necessarily trivial. However, as we approach the phase transition such behaviour disappears as instances become uniformly difficult. Visually, it is easy to observe that the three leftmost curves are straight, but too steep to be considered heavy-tailed, while the four rightmost curves are not sufficiently straight.

In Figure 1(b), we present results for the same problems, but with the variable ordering changed to min-dom/wdeg [4, 23]. Note that this change is sufficient to eliminate



(a) random variable and value orderings.



(b) min-dom/wdeg variable and random value orderings. Since chronological backtracking does no propagation, in this case min-dom/wdeg is effectively max-wdeg.

Fig. 1. The effect of the variable ordering on the complement of the CDF of backtracks in random problems using chronological backtracking. Problems instances are from a Model B generator [9]: 17 variables, 8 values, density 0.84, various tightness settings.

heavy tails from these problems since no straight line of appropriate slope can be observed for any tightness. Clearly, the change in variable ordering had a dramatic impact on the runtime distribution.

A common intuitive understanding of the extreme variability of the runtime of backtracking is that the search procedure sometimes must refute a very large inconsistent subtree, causing considerable “thrashing”. In our efforts to gain an understanding of this intuition, we consider how search ordering affects the runtime distribution of MAC, rather than chronological backtracking, as it is one of the most commonly used algorithms in constraint satisfaction. We study its runtime distributions over many instances of QWH-10 for several configurations of the algorithm. By changing the variable and value ordering heuristics used, we vary MAC’s “quality”, essentially creating different algorithms that we can use for our investigation. As mentioned earlier, we

used the following well-known *variable ordering heuristics*: min-domain [16], min-dom/ddeg [3], brelaz [5] and min-dom/wdeg [4, 23]. As *value ordering heuristics*, we used max-conflicts, random and min-conflicts.

In Figure 2 we present MAC’s runtime distributions when solving instances of QWH-10 with different variable and value ordering heuristics². When using a random value ordering heuristic, as presented in Figure 2(a), we observe heavy-tailed behaviour ($\alpha \in [0.579, 0.887]$) for every variable ordering heuristic studied. However, when we replace the value ordering with min-conflicts, as presented in Figure 2(b), we no longer observe heavy tails while using the min-dom/wdeg variable ordering, but straight lines ($\alpha \in [0.406, 0.685]$) can still be observed for all other heuristics³. For the min-dom/wdeg configuration we can see that the runtime distribution has become non-heavy-tailed, characterised by a curve in the log-log plot. Clearly, ordering heuristics also make a significant difference on this problem’s runtime distribution.

As we will show later in this paper, there are a number of factors at play that explain the effects that ordering heuristics have on heavy-tailed behaviour.

3 Definitions and Problems

Definition 1 (Binary Constraint Satisfaction Problem). We define a binary CSP as a 3-tuple $P \hat{=} \langle V, D, C \rangle$ where V is a finite set of n variables $V \hat{=} \{V_1, \dots, V_n\}$, D is a set of finite domains $D \hat{=} \{D(V_1), \dots, D(V_n)\}$ such that $D(V_i)$ is the finite set of possible values for V_i , and C is a finite set of constraints such that each $C_{ij} \in C$ is a subset of $D(V_i) \times D(V_j)$ specifying the combinations of values allowed between V_i and V_j , where $i < j$. We say that P is arc-consistent (AC) if $\forall v_k \in D(V_i)$ and $\forall j$ such that $C_{ij} \in C$, $\exists v_l \in D(V_j)$ with $(v_k, v_l) \in C_{ij}$. An assignment $A_{ik} \hat{=} \langle V_i = v_k \rangle$ represents a reduction of $D(V_i)$ to $\{v_k\} \subseteq D(V_i)$. A solution to P is a set of distinct assignments $\mathcal{S} \hat{=} \{A_{l_1 k_1}, \dots, A_{l_n k_n} \mid (v_{k_i}, v_{k_j}) \in C_{ij}\}$.

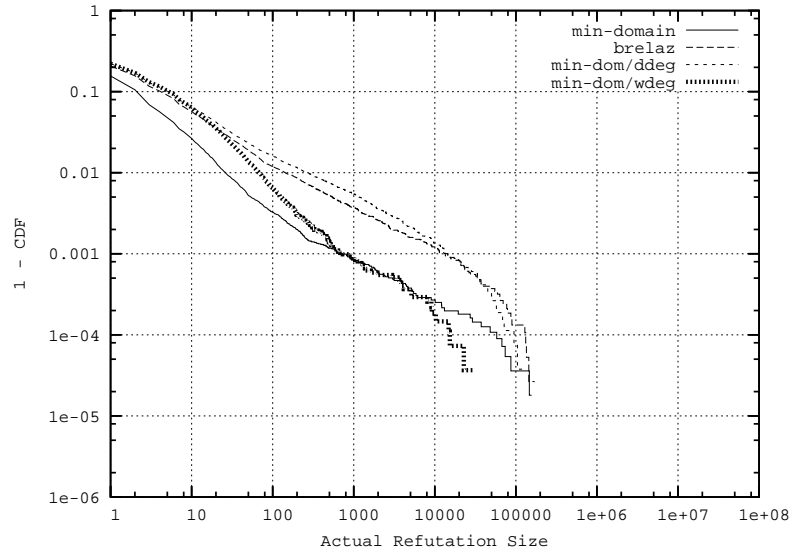
Definition 2 (Search Algorithm). A search algorithm $\Theta \hat{=} \langle \Lambda, \Delta, \prec_V, \prec_v \rangle$ is a combination of a branching method Λ , a consistency enforcement method Δ , a variable ordering \prec_V and a value ordering \prec_v , both of which can be either static or dynamic.

Definition 3 (Search Tree). A search tree \mathcal{T} for a problem P is a set of nodes and arcs. Each node corresponds to a set of assignments⁴, $\mathcal{N} \hat{=} \{A_{l_1 k_1}, \dots, A_{l_{p-1} k_{p-1}}, A_{l_p k_p}\}$, totally ordered by a variable ordering heuristic \prec_V . The root of the search tree is a special node $\mathcal{R} \hat{=} \emptyset$. Two nodes \mathcal{N}_1 and \mathcal{N}_2 are connected by an arc if $\exists A_{ij}$ such that $\mathcal{N}_2 = \mathcal{N}_1 \cup A_{ij}$, in which case we say that \mathcal{N}_1 is the parent of \mathcal{N}_2 , and \mathcal{N}_2 is the child of \mathcal{N}_1 . For every node \mathcal{N} , its children are totally ordered by a value ordering heuristic \prec_v .

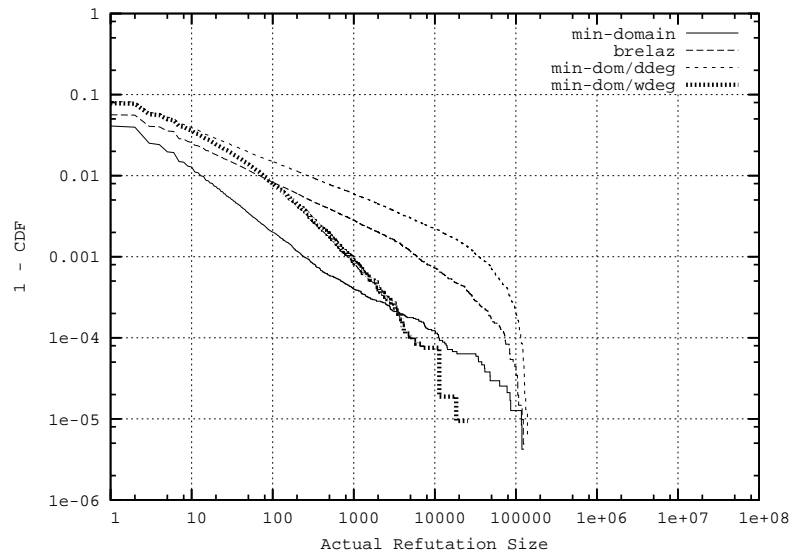
² In Figure 2, and in the remainder of the paper, we measure search effort in terms of cumulative “refutation sizes” for an instance, as this allows us to compare against search effort measured in terms of cumulative “optimal refutation sizes”. Refutations are introduced in Section 3.

³ A drop-off corresponding to less than 0.1% of the data is considered statistically insignificant [11].

⁴ For readability, the nodes in the example presented in Section 3.1 have been labelled only with the last assignment in the set.



(a) random value ordering heuristic and various variable orderings.



(b) min-conflicts value ordering heuristic and various variable orderings.

Fig. 2. Complement of the CDF of the cumulative effort required by MAC to solve instances of QWH-10 with different variable and value ordering heuristics. Note that when using a min-conflicts value ordering and a min-dom/wdeg variable ordering we no longer observe heavy tails (Figure 2(b)).

Search trees are defined in the context of a specific search algorithm. For a particular CSP instance P , and search algorithm Θ , a one-to-one mapping exists between the nodes in the search tree \mathcal{T} and the assignments made by Θ .

3.1 Mistake Points and Refutations

To study the effects that variable and value ordering heuristics have on heavy-tailedness, we focus on the refutations of the mistakes made by each algorithm studied.

Definition 4 (Mistake Point). *For a soluble problem P , a mistake point \mathcal{M} is a node identified by a set of assignments $\mathcal{M} \hat{=} \{A_{l_1 k_1}, \dots, A_{l_{p-1} k_{p-1}}, A_{l_p k_p}\}$, totally ordered by \prec_V , for which $\mathcal{M} \setminus \{A_{l_p k_p}\}$ can be extended to a solution, but \mathcal{M} cannot. Since an insoluble problem does not admit any solutions, we define the mistake point associated with an insoluble problem as the root of its search tree.*

Informally, a mistake point corresponds to an assignment that, given past assignments, cannot lead to a solution even though, in the case of a soluble problem, a solution exists. Whenever the value ordering heuristic makes such a mistake, the role of the variable ordering heuristic is to guide the search out of that insoluble search tree as quickly as possible. However, it is important to realise that the actual set of mistake points encountered during search is also dependent upon the variable ordering used.

For a soluble problem P , let $P_{\mathcal{M}} \hat{=} \{V_{\mathcal{M}}, D_{\mathcal{M}}, C_{\mathcal{M}}\}$ be the insoluble (sub)problem corresponding to \mathcal{M} , where $V_{\mathcal{M}} \hat{=} V \setminus \{V_{l_1}, \dots, V_{l_p}\}$, $C_{\mathcal{M}} \hat{=} \{C_{ij} | V_i, V_j \in V_{\mathcal{M}}, C_{ij} \in C\}$, and $D_{\mathcal{M}}$ is the set of current domains after arc-consistency has been restored to reflect the domain reductions due to \mathcal{M} . If P is insoluble, as a notational convenience, we define $\mathcal{M} \hat{=} \emptyset$ and $P_{\mathcal{M}}$ as the arc-consistent version of P . For brevity, we will refer to the *insoluble (sub)tree* rooted at a mistake point and its corresponding *insoluble (sub)problem* interchangeably.

Definition 5 (Refutations). *Given a search algorithm Θ , a **refutation** for a given insoluble (sub)problem $P_{\mathcal{M}}$, rooted at mistake point \mathcal{M} , is simply the corresponding search tree $\mathcal{T}_{\mathcal{M}}$. We will refer to $|\mathcal{T}_{\mathcal{M}}|$, the number of nodes in $\mathcal{T}_{\mathcal{M}}$, as the size of the refutation.*

We study the refutations found using a version of MAC for consistency enforcement and selects values randomly. Also, our version of MAC employs k-way branching [27], rather than binary branching, so that selecting a variable V_i creates $|D(V_i)|$ branches in the search tree. Our goal is to determine how close to optimality are the refutations obtained when well known variable ordering heuristics, with randomly broken ties, are incorporated as \prec_V . For each heuristic \prec_V , we first collect the mistake points it generates when using MAC (note that each variable ordering heuristic will generate a different set of mistake points). When we analyse MAC in conjunction with a certain \prec_V on a mistake point \mathcal{M} , we will refer to the refutation for the (sub)problem $P_{\mathcal{M}}$ as the *actual refutation*. We will contrast the actual refutation with the *optimal refutation* for $P_{\mathcal{M}}$, obtained by replacing \prec_V with a new variable ordering heuristic $\prec_{\overline{V}}$ such that $|\mathcal{T}_{\mathcal{M}}|$ is minimised.

To give a flavour of the kind of analysis reported in this paper, consider the example presented in Figures 3 and 4. Figure 3 shows MAC's [24] search tree for the 8-queens

problem. For simplicity, we used lexical variable and value ordering heuristics, although any other ordering, static or dynamic, would have worked as well. The five grey nodes in the tree are incorrect assignments, or mistakes – there are no paths below them that can lead to a solution, while such paths do exist for their parents. Each node is labeled with its corresponding assignment $V_i = v$, where V_i is an unassigned variable (row on the board) and v a value in its domain (column number). It is easy to see how the subtrees rooted at these mistake points make up the bulk of the search tree. Without them, the only thing the search would spend time on would be restoring arc-consistency after each correct assignment on the path to the solution.

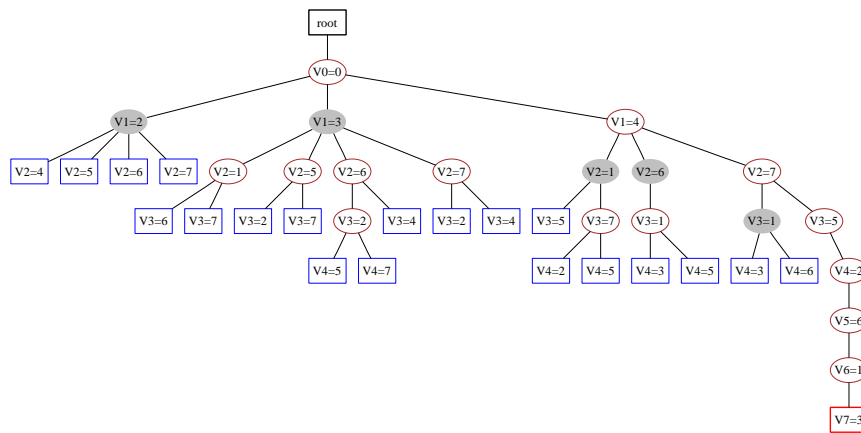


Fig. 3. MAC’s search tree for the 8-queens problem.

Once a mistake has been made, how quickly can the search recover from it? What is the minimum number of nodes required to prove, for instance, that the second insoluble tree in our example, the one rooted at node “ $V_1 = 3$ ”, is indeed insoluble?

By changing the order in which variables are selected, we can reduce the number of nodes required to prove insolubility. It can be easily verified⁵ that the tree in Figure 4 represents an alternative way of proving insolubility for the subtree corresponding to the second mistake. The alternative is made up of only 8 nodes, compared to the 14 in MAC’s original (actual) refutation.

It can also be verified, albeit with more difficulty, that the refutation in Figure 4 is optimal in terms of the number of nodes.

⁵ This is left as an exercise for the reader.

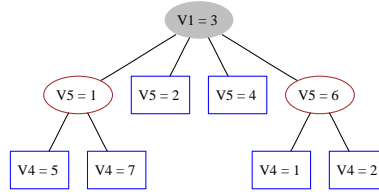


Fig. 4. Optimal refutation for the subtree rooted at “ $V_1 = 3$ ”.

Sometimes, when the optimal refutation is hard to find, we compute the *quasi-optimal refutation*, defined as the smallest refutation whose height does not exceed that of the actual refutation. Our experiments show that it is very rare that the quasi-optimal refutation is larger than the optimal (see Table 1). By accepting quasi-optimality, we can use the height of the actual refutation as an upper bound on the height of the optimal one, dramatically speeding-up the search for better refutations.

When searching for the quasi-optimal refutation it is oftentimes useful to compute a lower bound on its size. Firstly, this can improve the performance of the search since the search can be stopped when (and if) the smallest refutation found so far reaches that lower bound. Secondly, the lower bound can be plotted alongside the quasi-optimal refutation size to visually reduce the factor of uncertainty introduced by quasi-optimality. Optimal refutations may, in theory, be smaller than quasi-optimal refutations, but they cannot be smaller than the lower bound.

Definition 6 (Refutation Size Lower Bound). *An n -level lower bound for a refutation corresponding to a mistake point is the minimum number of nodes that an n -level look-ahead determines any refutation for that particular (sub)tree must have.*

A lower bound can be computed using the look-ahead method described in [18]. Such a lower bound is quite conservative, as there is absolutely no guarantee that an optimal refutation of that size exists, although that is often the case with QWH-10. We briefly review the look-ahead methodology here.

Whenever a variable V_i is selected at a certain level all the values in its domain have to be tried, and they all have to fail for the current (sub)problem to be proved insoluble. Consequently, we know that by selecting V_i , the size of the current partial refutation will increase by at least a number of nodes equal to $|D(V_i)|$. We call this a 1-level look-ahead.

By temporarily assigning to V_i , in turn, every value v in its domain, and by attempting to restore arc-consistency after every such assignment, we can associate with each v a minimum contribution to the size of the refutation. If the assignment makes the subproblem arc-inconsistent, v 's contribution will be 1, given by the node corresponding to the assignment itself. However, if arc-consistency can be restored after the assignment, at least one more variable will have to be considered before the current subproblem can be proved insoluble. Therefore, v will carry a minimum contribution equal to the small-

est domain size amongst all the remaining unassigned variables. We call this a 2-level look-ahead.

In general, V_i 's selection would increase the size of the current partial refutation by at least the sum of the minimum contributions of all the values in its domain.

Clearly, the further we look ahead, the less conservative the lower bound. However, look-ahead levels greater than 2 tend to be very time consuming and, therefore, the amount of look-ahead to use must be experimentally determined for each problem class being studied. We have concluded that for the class of quasigroup completion problems used in this paper, a look-ahead level of 3 was the most appropriate when computing the lower bound, and a look-ahead level of 1 was the most appropriate for the upper bound.

3.2 Problem Domain

As mentioned before, our experiments were focused around quasigroup completion problems. We briefly introduce them here.

Definition 7 (Quasigroup Completion Problems). *A quasigroup is a set Q with a binary operation $\star : Q \times Q \rightarrow Q$, such that for all a and b in Q there exist unique elements in Q such that $a \star x = b$ and $y \star a = b$. The cardinality of the set, $n = |Q|$, is called the order of the quasigroup.*

A quasigroup can be viewed as an $n \times n$ multiplication table defining a Latin square, which must be filled with unique integers on each row and column. The Quasigroup Completion Problem (QCP) is the problem of completing a partially filled Latin square. Quasigroup with holes (QWH) are satisfiable instances obtained by starting with a complete Latin square and unassigning a number of cells according to the Markov chain Monte Carlo approach proposed by Jacobson and Matthews in [20]. QWH problem instances are considerably harder when the distribution of the holes is balanced, i.e, when the number of unassigned cells is approximately the same across the different rows and columns [1, 20, 21].

Originally just mathematical curiosities, Latin squares have found practical applications in many scientific and engineering fields such as statistics, scheduling, drug tests design, and cryptography. Quasigroup problems have a small-world topology [28] and are known to be NP-complete [7]. QWH problems pre-assign a percentage of the cells in the table, introducing perturbations into the structure of the constraint network and bringing problems closer to real-world instances.

4 Finding Optimal Refutations

4.1 Summary of the Algorithm

We introduced in [18] an algorithm for obtaining optimal refutations for binary CSPs⁶. In this paper we significantly improved that algorithm's efficiency so that it can tackle

⁶ Code freely available with source at <http://hulubei.net/tudor/csp>, including code necessary to reproduce our experiments.

larger problems. All the optimisations described here are general in nature, i.e. they can be applied to any class of problem, but proved particularly useful in dealing with QWH-10 problems due to their relatively shallow optimal refutations. We observed that for the problems under consideration, while most actual refutations have heights up to 30, early experiments showed that most optimal refutations have heights below 5. The nature of the search for optimal refutations is such that the branching factor is significantly larger than that of a normal search tree, and we estimate that searching for an optimal refutation of height 6 for an instance of a QWH-10 problem could take several months to complete on a Pentium M CPU. Consequently, limiting the height of the refutations considered can dramatically improve efficiency.

An important design decision in the optimal refutation search algorithm was to use an iterative-deepening strategy [22]. The algorithm starts off by searching for refutations of height 1, then for refutations of height 2, and so on, until it reaches a height equal to either the number of variables in the (sub)problem, or the size of the smallest refutation found up until that point. The motivation behind using an iterative-deepening strategy is based on the expectation that small refutations are likely to have smaller heights than larger refutations, an expectation that has been validated by our experiments. The successive expansions of the search horizon can increase the likelihood of finding earlier refutations that are better than the actual, thus lowering the current refutation size upper bound and significantly speeding up the search in the rest of the tree.

Our improved algorithm uses the refutation size lower bound to limit the height in the iterative-deepening loop. Consider an example where we use m levels of look-ahead and obtained a lower bound of l nodes. The number of nodes in excess of m that would be part of any refutation of height m is $e = l - m$. If the best refutation found up until this point is of size r , then we can immediately conclude that since *any* refutation of height $d \geq m$ would contain at least e additional nodes, searching for refutations of heights greater than $r - e$ cannot produce a smaller refutation. Moreover, updating the lower bound as we search deeper into the tree allows us to stop the search as soon as it discovers a refutation whose size equals the lower bound (this seems to occur quite frequently). Finally, once the algorithm completes the search for refutations of size $r - e$, we know that the best refutation found is *optimal*.

The number of uninstantiated variables involved in each refutation is indirectly a factor affecting the height limit in the iterative-deepening loop. We have modified MAC so that, before selecting a new variable, it automatically marks as *instantiated* every variable whose domain has been reduced to a single value as a result of restoring arc-consistency. This reduces the height of the search tree, avoids unnecessary backtracking over singletons, and makes sure that such variables do not take part in any refutation. Insoluble trees thus involve a smaller number of variables and are easier to deal with.

4.2 Caching Optimal Refutations

An optimal refutation for a subproblem corresponding to a mistake point is made up of the optimal refutations of each one of the subproblems it is reduced to by the search algorithm. For large subproblems, the probability of one of their components occurring several times during search increases, in particular when problems are more structured.

One way of speeding up the search is to avoid looking several times for the optimal refutation of the same subproblem. Once the optimal refutation of a subproblem is found, it is stored for later retrieval in a hash table. The key used is the 16 bytes MD5⁷ raw digest of the textual representation of that subproblem concatenated with the depth limit in effect at that point in the search tree. The hash table associates the key with the search tree making up the corresponding optimal refutation. If the same subproblem is encountered later on during the search, its optimal refutation is quickly retrieved from the hash.

A detailed description of the caching mechanism is beyond the scope of this paper, but it is important to note that the depth limit imposed by the iterative deepening strategy makes a cached optimal refutation usable as-is only if the same subproblem occurs at a position in the search tree where the exact same depth limit applies. If a smaller depth limit is in effect, using the cached optimal refutation would cause the current, incomplete refutation to exceed its allowed depth. A greater depth limit would mean the possibility of further reducing the size of the cached entry still exists. In the former case, we can immediately deduce that no optimal refutation exists that satisfies that depth limit. In the latter, the size of the cached optimal refutation can be used as an upper bound. It is for this reason that the hash key had to include the depth limit in effect at the time the optimal refutation was discovered.

Experiments with QWH-10 show that for this particular type of problem caching makes a significant difference for difficult instances, i.e. those for which the search for optimal refutations proceeds with depth limits greater than 5. Depending on the problem instance, the speedup can be of an order of magnitude or more. However, for most difficult instances, caching exceeds 300-400Mb when the depth limit reaches 9, and clever management of the space allocated for caching becomes important. While we have not implemented any such strategies yet, ideas for future work include some type of LRU algorithm or caching only subproblems greater than a certain size.

4.3 Other Considerations

Despite of all these optimisations, our data-set contains a small number of instances (23 out of over 1,000,000) for which the search for the optimal refutation timed out after 2 hours for at least one mistake point, and in some of these cases no *improved* refutation was found. For some of these instances we succeeded in finding improved refutations by using a rapid random restarts strategy [14]. For others, we employed a certain level of *optimism*, i.e. we tried at each level a limited number of variables, in effect searching only what appeared to be the most promising area of the search space. The shorter-than-actual refutations that we found using these two methods are not known to be optimal or quasi-optimal, yet they are significantly smaller than their corresponding actual refutations, and by finding them we avoided incorrectly elongating the tails of the plots.

⁷ A standard cryptographic hash function.

5 Experiments

Our experiments were performed on satisfiable QWH-10 problem instances (90 variables) and on satisfiable QWH-20 problem instances (360 variables), both with 90% random balanced holes⁸, encoded as binary CSPs.

Our empirical study of QWH-10 included 4 variable ordering heuristics: *brelaz*, *min-domain*, *min-dom/ddeg* and *min-domain/wdeg*, and 3 value ordering heuristics: *random*, *min-conflicts*⁹ and its anti-heuristic, *max-conflicts*. We always broke ties randomly. Most instances were too difficult to solve using random variable orderings or variable ordering anti-heuristics, which is why these heuristics have not been included.

5.1 QWH-10 Instances

We aimed to study the relationship between actual and optimal refutations, as well as the way they evolve as better and better search algorithms are used to solve a large set of instances.

Using a Beowulf cluster of 32 CPUs over a period of 6 weeks we accumulated experimental data on all the 12 variations of MAC, totaling over 1,000,000 instances. Our intention was to avoid running an artificially randomised algorithm multiple times on the same instance. We computed actual refutations, lower bounds, and attempted to compute optimal refutations, reporting the cumulative size of each for every instance. While in the vast majority of cases we did find the optimal refutations, we encountered some instances for which we could only find improved refutations (i.e. refutations for which we were not able to guarantee quasi-optimality), and some for which the search timed out without finding any improved refutation. Table 1 gives the actual percentages.

Table 1. Optimality (%optimal / %quasi-optimal / %improved / %timed out).

	max-conflicts	random	min-conflicts
min-domain	99.37 / 0.19 / 0.40 / 0.03	99.44 / 0.25 / 0.31 / 0.00	97.96 / 0.74 / 1.28 / 0.02
min-dom/ddeg	95.41 / 4.01 / 0.58 / 0.01	98.27 / 0.42 / 1.30 / 0.01	86.67 / 3.37 / 9.84 / 0.11
brelaz	99.24 / 0.19 / 0.53 / 0.03	98.69 / 0.33 / 0.97 / 0.01	87.22 / 3.10 / 9.47 / 0.22
min-dom/wdeg	99.26 / 0.38 / 0.36 / 0.00	97.97 / 0.81 / 1.22 / 0.01	86.64 / 3.70 / 9.33 / 0.33

Figure 5 includes results for each of our 12 experiments and shows the effects of the various heuristics on the shape of the complement of the CDF for the cumulative effort required to solve each instance. The plots are organised roughly in increasing order of efficiency from left to right and from top to bottom. It is important to point out that the shorter refutation and lower bound plots are specific to each search algorithm,

⁸ Generated using code based on Carla Gomes' *lsencode* quasigroup generator.

⁹ This heuristic selects the value that is inconsistent with the smallest number of other values in the domains of neighbouring variables.

simply because different algorithms make mistakes in different places. The lower bound is plotted to give an absolute minimum on the size of the refutations even for those instances where we could not find the optimal or quasi-optimal refutations (see Table 1).

We notice from the first column of Figure 5 that a search algorithm employing a poor value ordering heuristic (max-conflicts) always exhibits heavy tails ($\alpha \in [0.567, 1.037]$), irrespective of which one of the 4 variable ordering heuristics we use. Moreover, heavy tails still exist, albeit with a different slope ($\alpha \in [1.328, 1.486]$ ¹⁰), even if, once a mistake has been made, the search algorithms were to use an oracle that could provide the shortest refutation for that mistake. In other words, in such cases the runtime distribution of an algorithm would be *guaranteed to exhibit heavy-tailed behaviour*.

We increase the quality of the value ordering heuristic by switching from max-conflicts to a random ordering and observe that while the distribution of actual refutations remains heavy-tailed, as we improve the quality of the variable ordering, the distributions of *shorter* refutations and lower bounds start becoming less and less heavy-tailed.

This phenomenon becomes even more pronounced as we move over to min-conflicts. In that case, the distributions of all lower bounds become non-heavy-tailed, and with the exception of min-domain, the distributions of shorter refutations become less heavy-tailed. Furthermore, the distributions of actual refutations become quite fat-tailed. Finally, min-conflicts + min-dom/wdeg succeeds at eliminating heavy tails even from the actual refutations, while at the same time keeping the actual refutations much closer to their corresponding optimal than any of the other 11 variations of MAC¹¹.

We observe in Figure 6 several factors contributing to the behaviour of our best performing algorithm. Firstly, for QWH-10, algorithms using the min-domain, min-dom/ddeg, and brelaz variable orderings encounter similarly large maximum refutations *regardless* of the value ordering used. Secondly, any algorithm using min-dom/wdeg encounters a maximum refutation size that is a factor of 5 smaller than the maximum encountered by algorithms using the other variable ordering heuristics. Thirdly, improving the value ordering heuristic from max-conflicts or random to min-conflicts results in a decrease in the probability of each non-trivial¹² refutation size occurring.

However, it is interesting to see in Figure 7 that any algorithm using min-dom/wdeg, while not encountering extremely large refutations, tends to have a higher probability of encountering all other sized non-trivial refutations, i.e. over the range of refutation sizes it encounters, it does worse than any other variable ordering for all value orderings.

Therefore, to summarise, the combination of the factors outlined above seems to eliminate heavy tails from the QWH-10 problems we have studied, despite the fact that neither min-conflicts nor min-dom/wdeg alone seems to be capable of doing that. This is a somewhat more complex scenario that one might have initially envisaged.

¹⁰ $\alpha \in (1, 2)$ denotes finite mean, but infinite variance.

¹¹ Table 1 shows that for min-conflicts + min-dom/wdeg we found the lowest percentage of optimal refutations, so the true runtime distribution can only be even more obviously non-heavy-tailed.

¹² Obviously, this implies an increase in the probability of the occurrence of trivial mistakes, i.e. those that can be refuted by propagation alone.

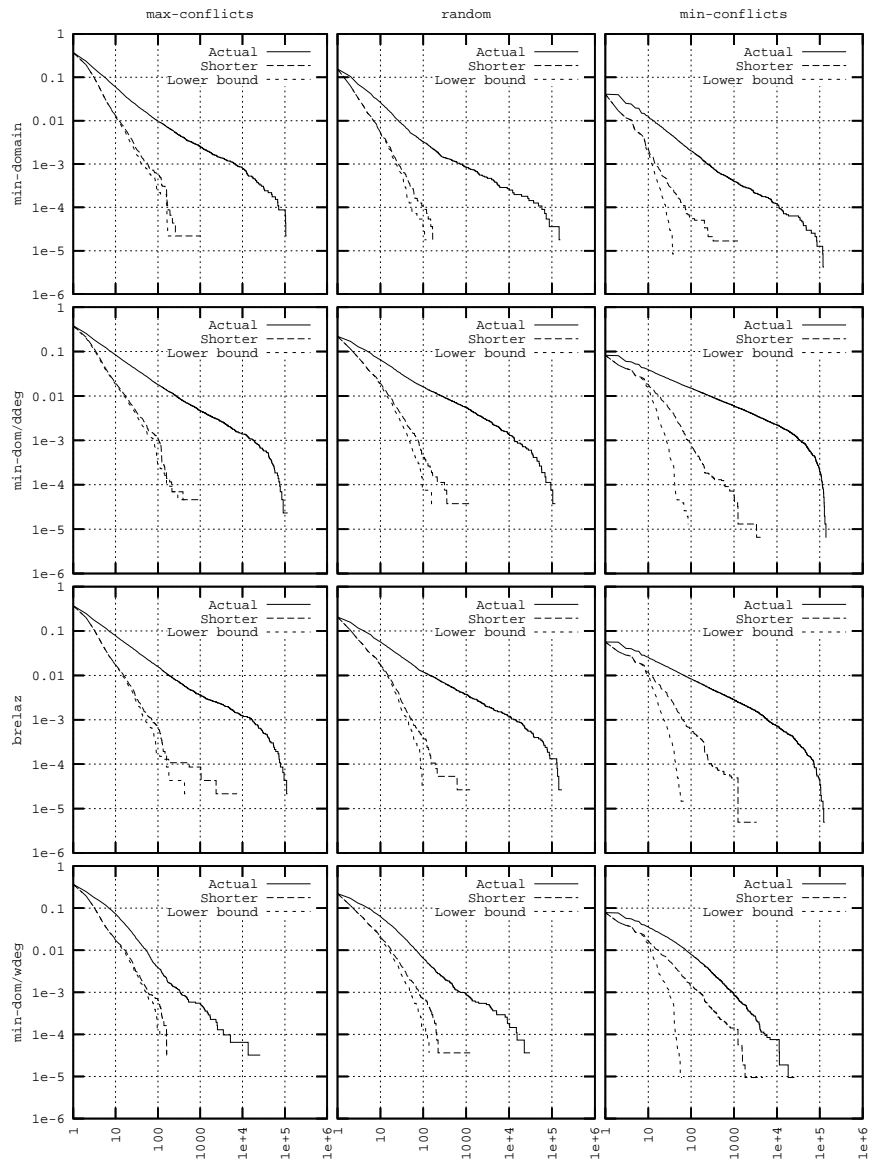
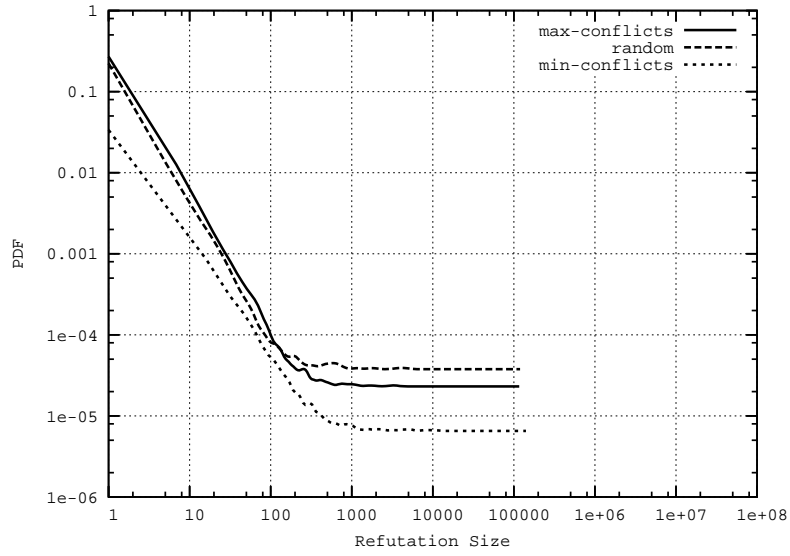
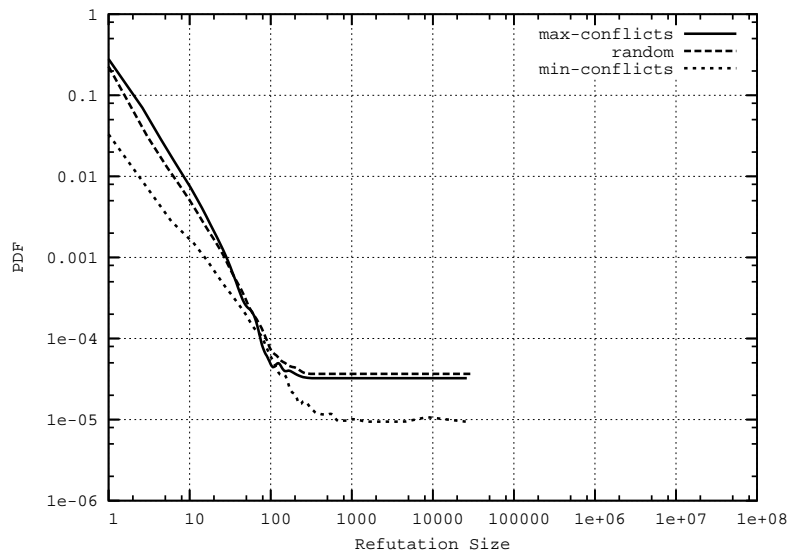


Fig. 5. Complement of the CDF of the cumulative effort required to solve instances of QWH-10 with 90% holes. We vary the value ordering across columns and variable ordering across rows. Shorter refutations are either optimal, quasi-optimal, or simply the shortest improved refutations we could find that were smaller than the corresponding actual refutations.



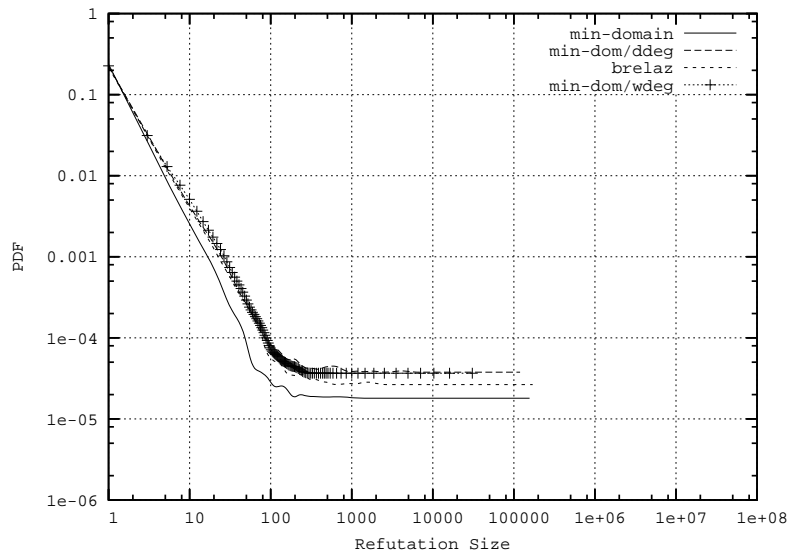
(a) Variable ordering: min-dom/ddeg (very similar results for min-domain and brelaz).



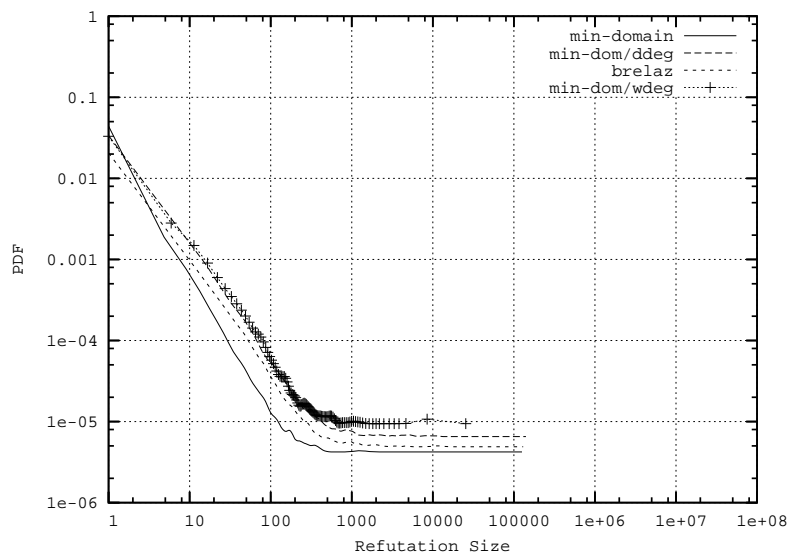
(b) Variable ordering: min-dom/wdeg.

Fig. 6. Probability distribution function for the actual refutation sizes obtained with various search algorithms, grouped by variable ordering.

The inherent heavy-tailedness that we observe suggests that while recovering from failure is important (which is typically the focus of research on variable orderings), we should also focus considerable attention on the interrelationship between variable



(a) Value ordering: random (very similar results for max-conflicts).



(b) Value ordering: min-conflicts.

Fig. 7. Probability distribution function for the actual refutation sizes obtained with various search algorithms, grouped by value ordering.

selection and value selection in order to mitigate heavy-tailed runtime distributions by failing less.

5.2 QWH-20 Instances

It is clear from the experiments described so far that the presence or absence of heavy tails is greatly influenced by the sophistication of the algorithm used. An interesting and legitimate question is what happens for larger problems? Would algorithms capable of eliminating heavy tails in QWH-10 scale and be able to eliminate heavy tails from larger problems, such as QWH-20?

To investigate this aspect, we have generated 500,000 satisfiable problem instances of QWH-20 with 90% random balanced holes (360 variables). We used min-dom/wdeg as a variable ordering and min-conflicts as a value ordering, i.e. the combination that eliminated heavy-tails from the distributions of actual refutations for QWH-10.

While a search for optimal refutations would be impractical for these problems as they are significantly more difficult than QWH-10, we have obtained actual refutations for all but 2187 instances (which timed out after 8 hours). Figure 8 shows that while heavy tails have not reoccurred, the tail of the plot has become significantly fatter, thus more linear and closer to what would be considered heavy-tailed, suggesting that our best performing algorithm may eventually be unable to eliminate heavy tails from much larger QWH problems. This result, however, says nothing about whether or not heavy tails will reappear when mistakes are refuted optimally. While based on the QWH-10 experiments we believe that this may be the case, further experiments will be required to firmly draw such a conclusion.

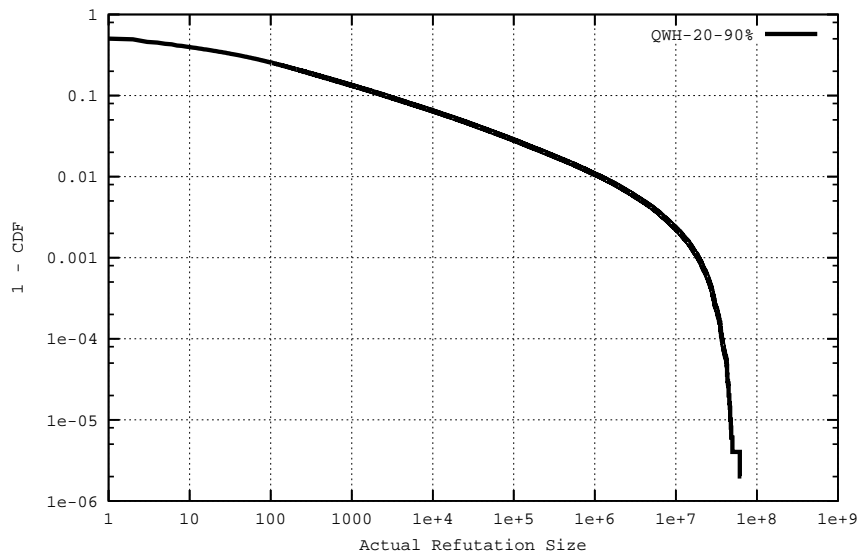


Fig. 8. Complement of the CDF of the cumulative effort required to solve instances of QWH-20 with 90% random balanced holes using min-dom/wdeg as a variable ordering and min-conflicts as a value ordering.

6 Conclusions

Much progress has been made on understanding problem hardness, typical-case complexity and the behaviour of backtrack search. The heavy-tailed phenomenon that characterises the runtime distributions of backtrack search procedures has received considerable attention. We have shown that a good choice of variable and value orderings can have a dramatic impact on the runtime distribution. A good combination can eliminate heavy-tailed behaviour from certain classes of problems, while a poor choice not only ensures that such behaviour is observed, but also that the nature of the insoluble subtrees encountered guarantees that this is the case.

We believe our work provides motivation for more focused research on the interplay between variable and value selection during search. We also believe that there are many directions that one can follow with respect to the utility of empirically studying optimal refutations of insoluble subtrees in relation to runtime distributions. Of particular interest would be the creation of a benchmark set of insoluble problems for which we know the bound on the size of their minimum refutations. We intend to explore these opportunities in more detail as part of our future work.

Acknowledgments

This paper is an extended version of [19]. This material is based on work supported by Science Foundation Ireland under Grant 00/PI.1/C075. We would like to thank Tom Carchrae, Mark Hennessy, Barbara Smith and Marc van Dongen, as well as our reviewers for their comments and suggestions, and to John Morrison and the Boole Centre for Research in Informatics for providing access to their Beowulf cluster. We thank Nic Wilson for suggesting that we consider caching when computing (quasi-)optimal refutations.

References

- [1] D. Achlioptas, C.P. Gomes, H.A. Kautz, and B. Selman. Generating satisfiable problem instances. In *Proceedings of AAAI-2000*, pages 256–261, 2000.
- [2] C. Bessière, C. Fernández, C.P. Gomes, and M. Valls. Pareto-like distributions in random binary csp. In *Proceedings of ACIA-2003*, 2004.
- [3] C. Bessière and J-C. Regin. MAC and combined heuristics: two reasons to forsake FC (and CBJ?) on hard problems. In *Proceedings of CP-1996*, LNCS 1118, pages 61–75, 1996.
- [4] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings of ECAI-2004*, pages 146–150, 2004.
- [5] D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.
- [6] H. Chen, C.P. Gomes, and B. Selman. Formal models of heavy-tailed behavior in combinatorial search. In *Proceedings of CP-2001*, pages 408–421, 1997.
- [7] C. Colbourn. Embedding partial steiner triple systems is NP-complete. *Combinatorial Theory*, A(35):100–105, 1983.
- [8] A. Davenport and E. Tsang. An empirical investigation into the exceptionally hard problems. Technical Report CSM-239, Department of Computer Science, University of Essex, U.K., 1995.

- [9] I.P. Gent, E. MacIntyre, P. Prosser, B.M. Smith, and T. Walsh. Random constraint satisfaction: Flaws and structure. *Constraints*, 6(4):345–372, 2001.
- [10] I.P. Gent and T. Walsh. Easy problems are sometimes hard. *Artificial Intelligence*, 70:335–345, 1994.
- [11] C.P. Gomes. Randomized backtrack search. *Constraint and Integer Programming*, pages 233–283, 2003.
- [12] C.P. Gomes, C. Fernández, B. Selman, and C. Bessière. Statistical regimes across constrainedness regions. In *Proceedings of CP-2004*, LNCS 3258, pages 32–46, 2004.
- [13] C.P. Gomes, C. Fernández, B. Selman, and C. Bessière. Statistical regimes across constrainedness regions. *Constraints*, 10(4):317–337, 2005.
- [14] C.P. Gomes, B. Selman, and N. Crato. Heavy-tailed distributions in combinatorial search. In *Proceedings of CP-1997*, pages 121–135, 1997.
- [15] C.P. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Automated Reasoning*, 24(1/2):67–100, 2000.
- [16] R.M. Haralick and G.L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14(3):263–313, 1980.
- [17] T. Hogg and C.P. Williams. The hardest constraint problems: A double phase transition. *Artificial Intelligence*, 69:359–377, 1994.
- [18] T. Hulubei and B. O’Sullivan. Optimal refutations for constraint satisfaction problems. In *Proceedings of IJCAI-2005*, pages 163–168, 2005.
- [19] T. Hulubei and B. O’Sullivan. Search heuristics and heavy-tailed behaviour. In *Proceedings of CP-2005*, pages 328–342, 2005.
- [20] M.T. Jacobson and P. Matthews. Generating uniformly distributed random latin squares. *Combinatorial Design*, 4:405–437, 1996.
- [21] H.A. Kautz, Y. Ruan, D. Achlioptas, C.P. Gomes, B. Selman, and M.E. Stickel. Balance and filtering in structured satisfiable problems. In *Proceedings of IJCAI-2001*, pages 351–358, 2001.
- [22] R.E. Korf. Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.
- [23] C. Lecoutre, F. Boussemart, and F. Hemery. Backjump-based techniques versus conflict-directed heuristics. In *Proceedings of ICTAI-2004*, pages 549–557, 2004.
- [24] D. Sabin and E.C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings of ECAI-1994*, pages 125–129, 1994.
- [25] B.M. Smith. In search of exceptionally difficult constraint satisfaction problems. In *Constraint Processing, Selected Papers*, LNCS 923, pages 139–156, 1995.
- [26] B.M. Smith and S. Grant. Sparse constraint graphs and exceptionally hard problems. In *Proceedings of IJCAI-1995*, pages 646–651, 1995.
- [27] B.M. Smith and P. Sturdy. An empirical investigation of value ordering for finding all solutions. In *Workshop on Modelling and Solving Problems with Constraints*, 2004.
- [28] T. Walsh. Search in a small world. In *Proceedings of IJCAI-1999*, pages 1172–1177, 1999.