A Grid-based Application of
Machine Learning to Model Generation

Volker Sorge^{1*}, Simon Colton^2, Andreas Meier^{3**}, and Roy McCasland^4

^{1} School of Computer Science, University of Birmingham, UK,
V.Sorge@cs.bham.ac.uk, http://www.cs.bham.ac.uk/~vxs
^{2} Department of Computing, Imperial College London, UK,
sgc@doc.ic.ac.uk, http://www.doc.ic.ac.uk/~sgc
^{3} DFKI GmbH, Saarbrücken, Germany,
ameier@dfki.de, http://www.ags.uni-sb.de/~ameier
^{4} School of Informatics, University of Edinburgh, UK
rmccasla@dai.ed.ac.uk, http://www.dai.ed.ac.uk/~rmccasla

Abstract. The classification of mathematical structures is a driving force in pure mathematics. A first step in producing algebraic classification theorems is to determine for which sizes certain algebras exist. Computational approaches to solving such existence problems using constraint satisfaction and model generation approaches have had much success. We look here at the question of distributing the model generation process using Grid technology. We present a novel distribution approach which involves using the HR machine learning program to intelligently suggest specialisations of the problem which are given to separate processors. Using the MACE, FINDER and SEM model generators, we demonstrate how this approach provides greater efficiency over a single-process approach for a series of quasigroup existence problems. We compare several approaches for the production and choice of specialisations, including the generation of proved classification theorems for algebraic structures of small sizes. We discuss how this approach could be used for more general problems.

1 Introduction

The classification of finite simple groups in 1980 was one of the major intellectual achievements of the twentieth century. Often, the first step towards such algebraic classification theorems is to determine how many structures exist up to isomorphism for each size. In particular, it is instructive to determine for which sizes algebras exist at all. Such existence problems have also attracted much attention in the research fields of automated reasoning and constraint solving. For instance, solving open quasigroup existence problems has become a challenge for friendly competition [18, 20], and such problems are used for benchmarking AI systems.[1] Existence problems are often solved by the construction of an algebra of the given size. If asked to perform the same task, a mathematician would probably not attempt to construct a multiplication table as a constraint solver or model generator would. Instead, they might determine the families which include a model of the given size, and attempt to write down the correct member of a particular family. For instance, writing down a multiplication table for a group of

---

* The author's work was supported by a Marie-Curie Grant from the European Union.
** Author's work supported by EU IHP grant Calculemus HPRN-CT-2000-00102.
[1] For example, see prob003 of the CSPLIB library (http://www.4c.ucc.ie/~tw/csplib).

size 17 is trivial, because only one group exists of this size and this group is cyclic, so its multiplication table consists of the seventeen possible cycles of the elements. Constructing a size 17 multiplication table from the axioms alone, however, is not trivial.

In the case of cyclic groups, a constructive theorem exists which describes how to build the models. The existence of such a theorem is not true in the general case. However, we can still propose a middle ground between the computational (brute force) approach and the more mathematical approach suggested above. This involves automatically specialising the model generation problem in the hope that one of the specialised – more constrained – sub-problems will be easier to solve. Such an approach may work if the existence problem can be solved positively, i.e., by actually finding a solution. The specialisations are generated via an un-supervised machine learning approach which uses smaller models that satisfy the axioms of the algebra to intelligently suggest properties to specialise the problem. For instance, if the problem was to produce a group of size 12, a model generator would be used to find groups of size 1 to 11, which become the input to an un-supervised machine learning system. The system would recognise that many of the models given to it are *Abelian*, i.e., $\forall x. \forall y. (x \circ y = y \circ x)$. Hence this suggests specialising the problem of finding a group of size 12 into the problem of finding an Abelian group of size 12, a much easier task.

We compare two machine learning approaches for automatically specialising model generation problems, including a sophisticated method that involves constructing and automatically proving (using automated theorem proving) classification trees for deciding the isomorphism family of algebras of given sizes. Both specialisation procedures are capable of producing large numbers of suitable sub-problems. This provides an opportunity for further efficiency gains by distributing the sub-problems over multiple processors. To gain the maximum efficiency possible, we have employed Grid technology to organise the distribution of processes. As a third specialisation method we also employed a simple method for distributing the problem, namely by instantiation variables. All specialisation methods are integrated into a uniform framework.

To demonstrate the effectiveness of our distribution approach, we performed a series of experiments using a standard problem set: QG-quasigroups. Using the MACE [15], FINDER [17] and SEM [21] model generators, we compared the three specialisation methods against a single-processor approach which uses the axioms alone. We show that in most cases, in particular, for larger sizes, the classification method for specialising into sub-problems produces substantial gains in efficiency over the other approaches, when distributed using Grid technology.

The paper is organised as follows. In section 2, we provide necessary background information, including a description of the problem domain (QG-quasigroups) and descriptions of the constituent AI programs in our system. In section 3, we provide an overview of how our system operates, including details of the Grid technology employed. In section 4, we provide details of the three methods employed to specialise the model generation problems. Following this, in section 5, we describe the experiments we performed in order to test the efficiency of the system and the results we obtained. Finally, in sections 6 and 7 we place this work in context and conclude by looking at the potential of this approach for more general problems.

2 Background

2.1 Problem Domain

Finite algebras describe a set of elements and a set of operators which each define a function taking a subset of elements to a boolean or a subset of elements, e.g., the multiplication function, which takes a pair of elements to a third element. The *axioms* of the algebra dictate properties of the operators which identify the algebra. Quasigroups – also known as Latin Squares – are algebras over a single multiplication operator, $\circ$, which is subject to a single axiom: $\forall a. \forall b. \ (\exists x. x \circ a = b) \land (\exists y. a \circ y = b)$. This corresponds to the property that every element appears in every row and every column of the multiplication table. As an example consider the following five quasigroups of order 3, which constitute representants of the five different isomorphism classes:

A_2	a b c
a	b a c
b	a c b
c	c b a

A_4	a b c
a	a c b
b	c b a
c	b a c

A_6	a b c
a	a b c
b	b c a
c	c a b

A_7	a b c
a	a b c
b	c a b
c	b c a

A_8	a b c
a	a c b
b	b a c
c	c b a

The discovery of quasigroups with certain properties is interesting outside of pure mathematics, because the underlying structure of quasigroups is similar to that found in many real-world applications, such as scheduling and timetabling, experimental design, error correcting codes, and wavelength routing in fibre-optic networks [12, 13]. Constraint satisfaction and model generating methods have been applied to decide quasigroup existence problems with much success (see section 6). Moreover, such problems have become benchmark domains for model generators and constraint solvers. Of particular interest are the so-called 'QG'-quasigroup families of problems [18] which algebras satisfying the quasigroup axiom and one of these additional axioms:

[QG1] $\forall a. \forall b. \forall c. \ (a \circ b) \circ b = (a \circ c) \circ c \rightarrow b = c$
[QG2] $\forall a. \forall b. \forall c. \ (b \circ a) \circ b = (c \circ a) \circ c \rightarrow b = c$
[QG3] $\forall a. \forall b. \ ((a \circ b) \circ (b \circ a) = a)$
[QG4] $\forall a. \forall b. \ ((b \circ a) \circ (a \circ b) = a)$
[QG5] $\forall a. \forall b. \ (((b \circ a) \circ b) \circ b = a)$
[QG6] $\forall a. \forall b. \ ((a \circ b) \circ b = a \circ (a \circ b))$
[QG7] $\forall a. \forall b. \ ((b \circ a) \circ b = a \circ (b \circ a))$

We are interested here in the following problem setting: given a specification of a (finite) algebra in the form of a set of axioms A, construct an algebra of given finite size n which satisfies these properties? Naturally, solving such problems also solves the question of whether an algebra of size n satisfying A exists.

2.2 Artificial Intelligence Systems Employed

The following Artificial Intelligence system were used in our experiments:
• The MACE [15] model generator first constructs all instances of the given axioms over the given finite domain and then employs a decision procedure that searches for models, i.e., satisfiable instances of the formulas.
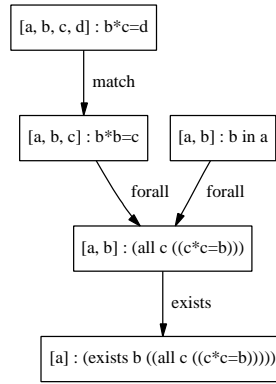
Fig. 1. Example construction by the HR System

- The SEM [21] model generator also first constructs all instances of the given axioms over the given finite domain. Then, it performs a heuristic search over the possible interpretations on the instances to find a model.
- The FINDER [17] model generator performs an exhaustive search for interpretations of the given axioms, using the axioms to direct its backtracking. Thereby, first a candidate model is generated and tried against all the axioms together. If all axioms are true, then the candidate model is accepted. If one of the axioms is false, the candidate model is adjusted to deal with the detected badness, resulting in a new candidate model.
- HR [1] is a machine learning program which performs automated theory formation by building new concepts from old ones using a set of production rules, and using measures of interestingness to drive the search [2]. It uses the examples of the concepts to empirically form conjectures relating their definitions and employs third party theorem proving and model generation software to prove/disprove the conjectures. In particular, for the application discussed in this paper HR used the following four production rules:

`Compose:` this composes functions using conjugation.
`Match:` this equates variables in predicate definitions.
`Exists:` this introduces existential quantification.
`Forall:` this introduces universal quantification.

As an example construction, consider the concept of there being a single element on the diagonal of the multiplication table of an algebra. This concept is constructed by HR using the match, forall and exists production rules, as depicted in Fig. 1. In this scenario, two concepts are supplied by the user, namely the concept of an element of the algebra and the multiplication of two elements to give a third. Using the match production rule with the multiplication concept, it invents the notion of multiplying an element by itself. By using this in the forall production rule, it invents the concept of elements which all other elements multiply by themselves to give. Then, using the exists production rule, HR invents the notion of algebras where there is such an element.

HR has a variety of different search strategies for employing the production rules. For the application described here, we employed a 'tiered' search for concepts, whereby the `Match`, `Exists` and `Forall` production rules were used greedily, with the compose rule used only when no further steps with the other rules were available. Al-

ternatively, HR can use heuristic searches whereby it employs a variety of measures of interestingness to assess the worth of a concept, then chooses the best concepts to build new concepts from [2]. For more details of how HR forms concepts, see [3]. In addition to forming concepts, HR makes conjectures about the concepts and uses third party software to prove/disprove these conjectures, details of which are described in [1], but are beyond the scope of this paper.

3   System Overview

In model generation, while solutions to small problem instances, e.g., QG5 quasigroups of size 5, are easily found, the search spaces explode combinatorially, and their sizes prohibit the systems from finding models for larger problems unless the problems are sufficiently constrained. Therefore, dividing the search space into more tractable parts is crucial for problems involving structures of a larger size. A division of the search space can be achieved by adding auxiliary properties to the problem in order to specialise the original axioms thus reducing the search space. In order not to change the problem itself it is necessary to divide the search into two parts: one with the additional property and another one with its negation. This ensures that the union of both search spaces will still contain all possible solutions.

When searching for a single model which satisfies the axioms, the hope is that it can be found faster in one of the two subspaces, than in the search space of the original problem. Thus, if two processors are available, parallelising the search into two separate model generation processes should produce a result faster than for the original problem. Furthermore, if we can employ a set of such specialising properties, a subsequent division into several parallel model generation processes should result in an overall speed up. The input to our system is a quadruple $\langle A, N, S, M \rangle$, where A is the set of axioms describing the algebra we are interested in, N is the order (size) of the algebra we want to construct a model for, S specifies the type of specialisation method to be employed and M is the model generator to use (SEM, MACE or FINDER).

Given this input, the system first generates a set of specialisation sub-problems P which consist of using model generator M to find models of size N which satisfy axioms A and an additional set of properties. The number and nature of the sub-problems depends on the specialisation process employed, as discussed in section 4. The specialisations used to attack the problem of, say, generating size 10 QG5-quasigroups can also be used for the problem of generating size 12 QG7-quasigroups. Hence, the specialisation procedure usually amounts to looking up the set of pre-determined specialisations to use, which have been generated automatically. Once the sub-problems P have been generated, for each P_i in P, model generator M is used to try to find a solution to P_i.

As discussed above, the benefit of the heuristic to constrain the model generation can be greatly improved if we distribute the search. Our approach lends itself well to distribution over a set of processors: each specialisation sub-problem can be run on a different processor. In order to achieve maximum distribution, we employ grid technology, which ensures that the processes are evenly distributed over a cluster of processors, as a single processor is assigned exclusively to each process. It also keeps the communication time to a minimum and free from interfering network traffic by

using the cluster specific interconnection facilities. The distribution itself is relatively straightforward: for each specialised axiom set a separate model generation process is created and distributed on the grid. When a process terminates, the result is checked, and if it has resulted in a solution, all other model generation processes are terminated immediately, since we have successfully solved the existence problem. If it does not contain a model, the other processes continue to run. Thus, if applied to a problem for which no model exists, all processes have to run until their search space is exhausted.

As discussed in section 5, we compare this approach with a single process approach that uses the original axiom set, hence covers the entire search space. This is run at the same time as the sub-problems, and is given the same status on the Grid as the specialisation processes. This has an additional benefit: for problems which have no solution, we have found that this process often finishes quicker than the set of specialisation sub-problems. Obviously, if the entire search space is traversed and no model is found, then no specialisation problem will be successful, and hence every process can be halted.

4   Generating Specialisation Concepts

An important process in our system is the way in which it decides how to specialise the original problem into a set of sub-problems for distribution. It is possible that adding properties to a set of axioms can increase the search space. In particular, if properties involve existentially quantified variables, the search for models of Skolem terms can enlarge the search space rather than reduce it. Even when the specialisation reduces the search space, we need to take into consideration the *constrainedness knife-edge* [9]: adding too few constraints can leave the problems unsolvable in a reasonable time, yet adding too many constraints may over-constrain the search space. Bearing these considerations in mind, we adopted an experimental approach where three methods for generating specialisations are compared and contrasted:

• One possibility is to split the search space into subspaces of roughly equal sizes, and we look at a simple way to do this in section 4.1.

• A second possibility is to more intelligently generate properties using machine learning procedures to generate and choose the properties. We discuss how the HR system can be used for this purpose in section 4.2.

• A final possibility is to attempt to place sub-problems onto the knife-edge, whereby the problem is constrained enough to be solvable in a reasonable time, yet not over constrained. We look at a method which uses classification trees to attempt to do this in section 4.3.

4.1   Instantiating Variables

As a simple way of specialising the model generation problems, we instantiated variables as follows: given a problem to generate an algebra of size n satisfying axioms A, the first specialisation has axioms A with the property that $0 * 0 = 0$, the second specialisation has axioms A with the property $0 * 0 = 1$ and so on, with the nth specialisation being axioms A with the property $0 * 0 = n - 1$. In addition, further specialisations with

properties $0 * 1 = x$ (with $x \in \{0, \ldots, n-1\}$) are possible. This acts as a control in our experiments – it may be that the distribution alone increases the efficiency, and not the specialisation procedures we have implemented, which would mean that this method would be competitive with the specialisation procedures.

4.2 Using Applicability

One approach to dealing with the constrainedness knife-edge is to be cautious, and choose properties which are not likely to over-constrain the search space. As discussed above, HR can generate numerous concepts which specialise the notion of algebra, and it can choose from these using various measures of interestingness [2]. To do this, the system takes the axioms of the algebra under investigation and uses the model generator specified to generate models of size 1, 2, 3, etc. up until the model generator fails to find a model in a given time. These are used as the objects of interest in a theory formation session with HR, and HR's search is adjusted to favour the production of specialisation concepts. In addition to working with smaller algebras satisfying exactly the same axioms as the problem we are looking at, we can employ more general or more specific axiom sets for property generation.

Once the theory has been produced, the *applicability* measure is used to extract the best specialisations with respect to that measure. The applicability of a concept is measured as the proportion of models which satisfy the concept. The idea is that, when being cautious, it is better to choose specialisations with high applicability, as, given the evidence of smaller sizes, it is unlikely that these will over-constrain the search space. In practice, applicability is used via a threshold: given a theory formed by HR which contains specialisation properties $\{P_1, \ldots, P_m\}$, the system chooses a subset of these concepts as possible constraining properties by considering those concepts that hold for at least a certain, specified number of the given structures. Given a threshold $0 < t \leq 1$, only those properties are considered that actually hold for at least $t \cdot n$ structures.

For instance, if $t = 0.5$ all properties are chosen that hold for at least half the models. The system then takes the set of chosen properties $\{Q_1, \ldots, Q_l\} \subseteq \{P_1, \ldots, P_m\}$ and generates sets of axioms that will be employed to generate models. Suppose we are looking for structures that satisfy a particular set of axioms $\mathcal{H}$, the system will then generate the sets of axioms of the form $\mathcal{H} \cup \{Q_1\}$, $\mathcal{H} \cup \{Q_2\}$, ..., $\mathcal{H} \cup \{Q_l\}$, and $\mathcal{H} \cup \{\neg Q_1 \vee \neg Q_2 \vee \ldots \vee \neg Q_l\}$. These $l + 1$ axiom sets are then submitted to the model generators. Any structure found will naturally also be a model of the original axiom set $\mathcal{H}$. As an example, when considering existence problems of QG5 quasi-groups, to generate properties with HR, the system generated a set of QG5 quasigroups of size 3 to 9. HR then came up with 68 properties of which 19 were chosen as they hold for more than half of the quasigroups. These resulted in 20 sub-problems being given to the model generators.

4.3 Using a Classification Algorithm

Being cautious with the addition of specialising properties may mean that the sub-problems are still under-constrained and take too long to solve. One way to reduce the search space without over-constraining is to reduce symmetry: two solutions which

may be syntactically different, may be considered by the user as really the same, and adding constraints which rules out one of these solutions without ruling out the other is said to break the symmetry. In algebraic domains, the main cause of symmetry is *isomorphism*. Two algebras are considered isomorphic if a permutation of the elements of one means that it can be re-written to look exactly like the other one. Hence, when generating algebras, we could consider a problem on the constrainedness knife-edge if it was guaranteed to find only solutions in a particular isomorphism class. Some computationally tractable techniques exist for determining isomorphism types – e.g., for groups [16] – but in general, it is as difficult to do this as it is to exhaust the space of models for a given size. However, we can approximate the properties required for isomorphic specialisation of the problem size of interest, by looking at properties which can be used to classify smaller sized algebras up to isomorphism.

In [4], we present a bootstrapping algorithm for generating theorems which enable the classification of algebras up to isomorphism. As a simple example, the algorithm is given the axioms of group theory and told to find a classification theorem for groups of size 6. It returns the following (paraphrased) result: "all groups of size 6 can be classified up to isomorphism as either Abelian or non-Abelian". The implementation of this algorithm uses HR, MACE, SEM, FINDER, the automated theorem prover Spass, and the Gap computer algebra system [8] to generate such results. Note that Spass is used to show that the theorems provide valid classifications by proving that each concept in the theorem is a *classifying property*, i.e., true for all members of exactly *one* isomorphism class.

The general idea of the algorithm is to construct a decision tree by successively generating non-isomorphic algebraic structures and associated discriminants until all isomorphism classes have been found. A discriminant property P for two non-isomorphic structures A_1 and A_2 is a property that is invariant under isomorphism such that P does hold for A_1 but not for A_2 or vice versa. That is, the existence of P for A_1 and A_2 is a proof that A_1 and A_2 are not isomorphic. The algorithm takes the size n and the axioms of the algebraic structures to be considered as input. It returns a decision tree for isomorphism classes of the algebraic structure that serves as classification for the specified algebras up to isomorphism for a given cardinality.

For example, Fig. 2 gives the decision tree constructed for the classification of quasigroups of order 3. The edges of the tree are labelled with properties. Each node in the tree is associated with the conjunction of the properties that are the labels of the edges of the path leading from the root node to the node. For instance, node 4 is associated with the property $P_1 \wedge P_2$. The tree shows 5 isomorphism classes for quasigroups of size 3 corresponding to the leaf nodes 2, 4, 6, 7, and 8, which are called the isomorphy nodes of the tree. For isomorphy nodes, the property associated with the node uniquely characterises an isomorphy class of quasigroups of size 3. That is, all quasigroups of size 3 satisfying the property associated with an isomorphy node are isomorphic. The representants of the isomorphism classes are the quasigroups given in section 2.1 where the enumeration of the quasigroups corresponds to the labels of the respective isomorphy nodes, e.g., A_2 is the representant of the isomorphism class given by node 2.

Initially, the decision tree consists only of the root node. The single nodes of the tree are expanded by first generating an algebraic structure satisfying the properties specified by the node. For example, for the root node 1 in Fig. 2, an arbitrary quasigroup

of order 3 is constructed; for node 3, however, a quasigroup that additionally satisfies the property $\exists b.\, b \circ b = b$ is required. If no representant can be generated, we must prove that there exists no algebraic system of cardinality n satisfying both the original axioms and the additional properties generated up to this point.

When a representant is generated we have two cases to consider: there exists a non-isomorphic structure exhibiting the same properties considered so far or the property represented by the node constrains the structures to a single isomorphism class. In the former case, we have to branch further by generating a structure non-isomorphic to the original one. The two structures are then passed to HR to compute two — generally the smallest — discriminants. In the example, the quasigroup A_4 given in section 2.1 was constructed as a non-isomorphic counterpart to A_2. Given those two quasigroups, HR came up with discriminants P_1 and $\neg P_1$ as stated in Fig. 2. Depending on the nature of the discriminants, either two or four child nodes are constructed. The case of two child nodes can be observed in the expansion of nodes 1 and 3 in Fig. 2, whereas the expansion of node 5 leads to four children.

This construction mechanism guarantees that the resulting tree is a decision tree for algebras, i.e., each single branching as well as the whole tree are covering (an algebra satisfies the properties associated with *at least* one child node/leaf node) and exclusive (an algebra satisfies the properties associated with *at most* one child node/leaf node). However, by this construction mechanism the algorithm can also introduce nodes for which no algebra of the considered size and axioms exists. Such nodes are called *dead-ends*. For instance, since there is no quasigroup of size 3 satisfying the properties P_1, $\neg P_2$, and $\neg P_3 \wedge \neg P_4$ node 9, in Fig. 2 is a dead-end node.

The tree in Fig. 2 is a classifying decision tree for quasigroups of size 3. That it is a decision tree, however, holds for arbitrary algebras, for instance, also for quasigroups of size 4, 5, 6, …. However, for other algebras the tree is not longer classifying, i.e., the leaf nodes do not necessarily specify isomorphy classes and dead-ends for other algebras.
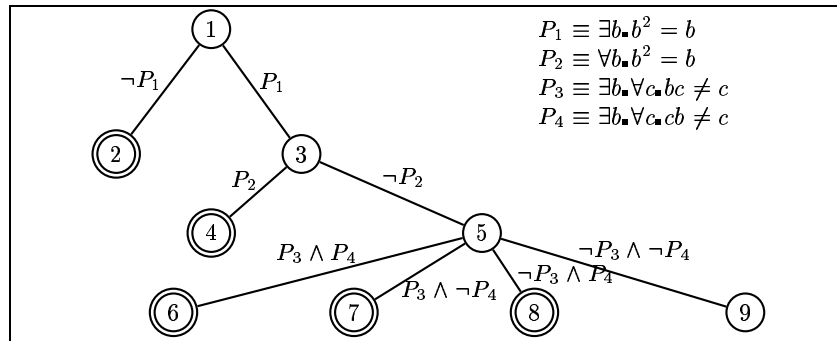

Fig. 2. Decision tree for the classification problem of order 3 quasigroups.

The classification trees produced by this algorithm are employed by the system discussed in this paper as follows. Let $\{P_1, \ldots, P_n\}$ be the properties associated with the leaf-nodes of a classifying decision tree and let $\mathcal{H}$ be a set of axioms, then our

system combines these to n specialisations of $\mathcal{H}$ by adding each property to the set of original axioms. These sets of axioms are then submitted to the model generators. For instance, consider the classifying decision tree for quasigroups of order 3 depicted in Fig. 2. The specialisation properties are the properties associated with the leaf nodes. We therefore get six different axiom sets as specialisations: $\mathcal{H} \cup \{\neg P_1\}$, $\mathcal{H} \cup \{P_1 \wedge P_2\}$, $\mathcal{H} \cup \{P_1 \wedge \neg P_2 \wedge P_3 \wedge P_4\}$, $\mathcal{H} \cup \{P_1 \wedge \neg P_2 \wedge P_3 \wedge \neg P_4\}$, $\mathcal{H} \cup \{P_1 \wedge \neg P_2 \wedge \neg P_3 \wedge P_4\}$, and $\mathcal{H} \cup \{P_1 \wedge \neg P_2 \wedge \neg P_3 \wedge \neg P_4\}$.

Note that this heuristic can be approximated by the applicability approach described in section 4.2. If the system started off with a covering set of non-isomorphic algebras for each size, and if the threshold was set so that properties which applied to only a single algebra were chosen, then each property must be a classifying property (although, as Spass is not employed, this is not proved). However, in practice, there is no guarantee that the model generator has produced exemplars of each isomorphism class, and when the number of initial algebras becomes more than around 10 or so, HR usually fails to produce classifying properties for all isomorphism classes. In addition to potentially placing the sub-problems closer to the constrainedness knife-edge, another potential advantage of the classification tree approach is that, since the specialisation properties result from a decision tree, they divide the search space of possible algebras disjointly and hence there is little repetition of work. Also, it is more likely that the specialisation properties split the search space more evenly. Moreover, the properties cover the search space, so no solutions are lost.

5   Experiments and Results

Our aim here is to compare and contrast the three specialisation methods described in the previous section. In particular, we want to determine whether any gains in efficiency can be made when applying model generators FINDER, SEM and MACE to the quasigroup existence problems introduced in section 2.1. We have used each specialisation technique for all of the QG problems, but, due to space considerations, we present here only the results for QG1, QG4 and QG5 (for more detailed results please see ftp://ftp.cs.bham.ac.uk/pub/authors/V.Sorge/quasigroups/results.dvi). We compared the seven following approaches to solving the model generation problems:

• [Orig] This used a single processor with the original (non-specialised) axiom set. This is essentially a benchmark our parallel approach with respect to the performance of the original systems.

• [Inst1 + Inst2] refers to the two methods described in section 4.1. For Inst1, we fixed a single pair of elements $(0, 0)$ and pre-instantiated all possible results. For a problem involving quasigroups of size n the heuristic resulted in n specialised axiom sets. For Inst2, we additionally fixed a second pair $(0, 1)$, which resulted in $n \cdot (n - 1)$ specialisations.

• [Appl] refers to the method given in section 4.2. The 19 concepts of applicability 0.5 or higher resulted in 20 specialised axiom sets.

• [Class3, Class4, Class5] employ results of the classification algorithm described in section 4.3. In detail, Class3 uses the classifying properties from the decision tree for

System	Problem	Orig	Inst1	Inst2	Appl	Class3	Class4	Class5
FINDER	QG5.6*	0.04	0.03	0.08	0.05	0.05	0.06	0.15
	QG5.7	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	QG5.8	27.06	0.04	0.01	27.27	0.01	0.02	0.02
	QG5.9	0.04	0.02	0.02	0.01	0.04	0.04	0.01
	QG5.10*	3569.51	35300.13	7905.71	t/o	3570.17	t/o	t/o
	QG5.11	t/o	62.36	9.01	t/o	21.84	16.12	15.25
	QG5.12	316.42	317.22	57.26	332.12	321.72	3.57	494.4
	QG5.13	11342.30	12881.42	1682.91	8217.31	4014.80	3533.37	118.74
SEM	QG5.6*	0.00	0.00	0.01	0.01	0.00	13.71	142.55
	QG5.7	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	QG5.8	0.06	0.11	0.01	0.02	0.00	0.00	0.00
	QG5.9	0.00	0.00	0.00	0.00	0.02	0.01	0.00
	QG5.10*	1172.52	t/o	42749.04	1927.82	15193.60	t/o	t/o
	QG5.11	t/o	t/o	19.19	2518.37	38.69	40.28	170.89
	QG5.12	t/o	84.72	t/o	t/o	3.94	0.01	0.10
	QG5.13	t/o	t/o	t/o	t/o	t/o	t/o	89.83
MACE	QG5.6*	0.03	0.05	0.12	0.06	0.01	8.38	5.31
	QG5.7	0.57	0.58	0.60	0.01	0.01	0.01	0.01
	QG5.8	46.80	47.40	48.75	0.58	0.57	0.58	0.46
	QG5.9	48.52	48.63	47.95	0.00	0.00	0.01	0.01
	QG5.10*	t/o	t/o	t/o	t/o	t/o	t/o	t/o
	QG5.11	t/o	t/o	t/o	t/o	t/o	t/o	t/o
	QG5.12	t/o	t/o	t/o	t/o	t/o	105.04	t/o
	QG5.13	t/o	t/o	t/o	t/o	t/o	t/o	t/o
FINDER	QG4.6*	1.58	0.33	0.04	1.58	1.22	1.20	2.07
	QG4.7*	84.75	38.77	8.81	98.26	89.87	90.59	123.47
	QG4.8	0.05	0.02	0.03	0.05	0.06	0.06	0.61
	QG4.9	0.38	0.14	0.01	0.31	0.37	0.36	0.66
SEM	QG4.6*	0.00	0.01	0.02	0.12	0.01	45.29	29.00
	QG4.7*	0.06	0.63	0.97	0.21	0.16	885.64	488.59
	QG4.8	0.09	0.02	0.00	0.01	0.01	0.01	2.26
	QG4.9	2.51	0.22	0.01	0.03	0.01	0.01	0.05
MACE	QG4.6*	0.07	0.07	0.13	0.10	0.02	37.82	54.66
	QG4.7*	2.64	2.87	5.68	4.22	0.31	76.78	4123.43
	QG4.8	61.49	61.45	60.86	0.88	0.87	0.92	2.27
	QG4.9	27023.21	26798.08	27040.67	2343.61	329.87	339.73	134.94
FINDER	QG1.6*	1660.59	782.67	117.88	1885.42	4875.55	5139.38	t/o
	QG1.7	0.34	0.26	0.24	0.05	0.09	0.01	0.02
	QG1.8	12776.18	100.18	4778.38	2.50	8704.15	4.54	74.13
	QG1.9	6438.08	74275.36	144.66	8929.90	6518.62	7.82	82.71
SEM	QG1.6*	0.69	6.58	1.36	2.16	1.62	6313.57	t/o
	QG1.7	0.29	0.01	0.00	0.01	0.01	0.00	0.00
	QG1.8	1566.77	0.34	0.37	0.04	1.25	0.01	0.21
	QG1.9	0.00	0.00	0.00	0.00	12.49	0.00	0.00

Fig. 3. Experimental results for the QG problem domain. Times are given in seconds, t/o indicates the model generator exceeded the 1 day timeout. Asterisks indicate that a given problem does not have a solution.

quasigroups of order 3, which comprises 6 properties and therefore the same number of specialisations. Class 4 uses the 41 properties of quasigroups of order 4 produced for the decision tree. For quasigroups of order 5, the decision tree yields more than 1500 properties, which we considered an impractical size. We therefore used the data from an intermediate decision tree for order 5 quasigroups, which yielded 468 properties, i.e., the tree still had all the properties of a decision tree, but the leaf nodes were not necessarily isomorphism classes for quasigroups of size 5.

The results from running these methods for QG1, QG4 and QG5 quasigroups of various sizes are given in table 5. Each row contains the time (in seconds) needed to solve a problem. The first column indicates the model generator for which the results were obtained; the second column specifies the particular problem, where QG$m.n$ is the problem to show the existence of an order n quasigroup for which QGm holds. Asterisks indicate that a given problem does not have a solution. For example QG5.6* is the problem to find a QG5 quasigroup of order 6 and the asterisk indicates that there is no such quasigroup. T/o indicates the model generator exceeded a 1 day timeout. The times are recorded as the time when the first of the parallel model generation pro-

cesses succeeded in finding a model. In the cases where no model exists, all processes have to terminate in order to guarantee that the search was exhaustive. Thus the time corresponds to the run time of the last process that terminated.

The experiments where conducted on a 113 node 2GHz Pentium 4 cluster with 2GB of RAM each with regular Ethernet interconnect. As a grid engine, we employed GridEngine v5.3 running on a quadruple 2.4GHz Xeon with 4GB of memory. For some of the experiments – notably for Class5 and the Inst2 heuristic for quasigroups of order > 10 – the number of specialisations exceeded the capacity of the cluster. We therefore ran these in several blocks, and the results should be understood as a simulation of a cluster of the necessary size. The results for the problems for which no model exists (QG5.6, QG5.11, QG4.6, QG4.7, QG1.6) demonstrate that in this case our specification approach fails. This is not surprising, as to conclude that no model exists, the search must terminate on all specialised axiom sets. Thus, if only one of the added properties has a negative effect on the search space, it affects the overall result.

The situation is exactly the opposite if there exists a model. Then, one specialised sub-problem which provides a model is sufficient to solve the overall problem. The results of our experiments demonstrate that employing the instantiation heuristic already gives a speed up in the cases where a model exists. Employing the classifiers can often further increase the speed up. In particular, for the more complex problems, the speed up is sometimes not only significant but also the model generators can find solutions for problems on which they timeout without the specialisations.

The results for the problems for which a model exists do not point out a clear winner, i.e., no method consistently outperformed all the other methods. However, altogether for each problem for which a model exists and which is non-trivial (i.e., the original problem is solved in less than 1 second) at least one specialisation heuristic gives a considerable speed up.

Another pattern also appears to emerge: for the small sizes, there seems to be little point in using a specialisation method. For the intermediate sizes, simply distributing the process using the Inst2 method appears to be quicker in many instances. For the larger sizes, however, the classification methods appear to do better. Also apparent is that the classification methods usually outperform the applicability method. It is very interesting to compare the three decision trees methods for the generation of specification sub-problems. The Class4 decision tree outperforms the Class3 decision tree on almost all problems for which a model exists (and for the exceptions of this rule the Class3 tree performs only slightly better). However, for the Class5 tree we cannot observe a clear improvement as compared with the Class4 tree: except for QG5.13 it does not perform significantly better but on the other problems it performs worse. This may indicate that our approach is limited: splitting the problem into more sub-problems will not necessarily result in faster solutions. We intend to evaluate this hypothesis with more experimentation.

6 Related Work

The study of quasigroups, and more generally, automated discovery in finite algebras, stimulated much research in Automated Reasoning. For instance, [7, 14, 18, 19] report

on the use of model generation techniques and propositional provers to tackle quasi-group existence problems. In particular, open problems in quasigroups became a challenge for friendly competition. J. Zhang was the first to use a general reasoning program to solve an open quasigroup existence problem. Later, Fujita, Slaney, Stickel, McCune, and H. Zhang used their systems to solve several open cases and reported very competitive results. More recently, completing partial quasigroups has been proposed as a structured benchmark domain for the study of constraint satisfaction methods [11].

The difficulty with using automated reasoning techniques and constraint satisfaction techniques is that the underlying problems are intractable (undecidable, NP-complete). No matter how good the algorithms are, there is a constant need for more computing power. This motivated the distribution of problems and the application of concurrently running applications in the context of quasigroup existence problems. [19] presents Psato a parallel prover for propositional satisfiability for networks of workstations. A key property of the approach is that the concurrent processes explore disjoint portions of the search space such that parallelism is employed without introducing redundant search. [11] and [10] report on the possible concurrent application of randomised algorithms, which are among the best current algorithms for solving computationally hard problems. The time required by these algorithms to solve a particular problem instance can vary drastically (indeed the algorithms studied in [11, 10] exhibit so-called heavy-tailed runtime distributions). Hence, running several instances of a randomised algorithm can boost the performance.

In contrast to Psato, the randomised algorithms all search the same search space but because of the inherent randomisation they traverse it differently. Our approach is similar to the Psato approach with respect to the disjoint partition of the algebra space that results from the usage of decision trees. However, since the generation of classifying decision trees is not deterministic (see [4] for details) we could also randomly create different decision trees to use in our approach and employ concurrency on these decision trees, which would then be similar to different runs of randomised algorithms.

Automatically re-formulating problem statements in order for solvers to more efficiently find solutions has been addressed in many areas of Artificial Intelligence, most notably constraint solving. In particular, the process of deriving additional information from the problem statement so that *implied constraints* can be added without loss of generality has been researched recently [6]. Also, in [5], Colton and Miguel use the HR program to generate additional constraints for quasigroup existence problems, and this approach provided up to 10 times speed up for certain problems. In contrast to the work described here, however, this approach was semi-automated, ad-hoc and concentrated more on using implied constraints, rather than the case splits we use here. It also focused on improving a single constraint solver, rather than distributing the workload over a grid, as discussed here.

7 Conclusions and Further Work

We have presented a novel approach to distributing a model generation process for solving algebraic existence problems. Based on a general framework, we investigated several methods to split a model generation problem into multiple more specific sub-

problems, which can be processed in parallel. The specialisations range from simple instantiation to more intelligent choosing of specialisation properties. Our experiments demonstrate that, by using a grid-based concurrent distribution, our approach provides greater efficiency over a single-process approach. Moreover, in many cases, the more elaborate specification approaches outperformed the simple specification approaches. We plan to perform more experiments in order to determine more precisely when a particular approach will outperform another.

We have used this approach for the QG5 problem of size 18, which is still an open problem. Unfortunately, none of the specialised sub-problems has returned a solution, which gives some evidence towards the hypothesis that no solution exists. We have also looked for cases where a search using specialising property P and a search using specialising property $\neg P$ have both ended unsuccessfully, thus proving that there are no solutions, but none have been forthcoming. One avenue for further work may be to generalise this approach: suppose that the search for sub-problems with specialisations $P_1, \ldots, P_n$ have all finished without solutions. Then, if a solution does exist, it must have the property $\neg P1 \wedge \ldots \wedge \neg Pn$. Hence, we could take successively larger subsets of $\{\neg P1, \ldots, \neg P_n\}$ and attempt to prove with an automated theorem prover that all the properties in the subset holding at once is inconsistent with the axioms, hence showing that no solution can exist.

Our experiments with the three specialisation methods described in the paper did not provide us with a clear winner, i.e., a method that outperforms all other methods on all problems. Furthermore, without additional experimentation, it is difficult to draw generalisations about how best to specialise model generation problems into sub-problems. However, the specialisation methods showed a significant speed up over a single-process approach, and, in particular, for larger sizes, the classification methods appear to be better than the other specialisation methods. It is also difficult to choose a clear winner for the the three different classification trees approaches. In the worst case, a specialisation method turned out to take even longer than the original problem. These observations raise the question whether it is possible to predict the performance of a particular method on a particular problem. If this is possible, then a suitable method can be chosen. Our current experiments do not provide sufficient data to give an account on this question. However, we hope that further experiments will provide enough data that we can try to figure out (learn) problem and specification parameters for an automated selection of the specification method.

Although we experimented here only with quasigroups, our approach is general enough for model generation of any finite algebra with particular properties. Moreover, the idea to specialise problems goes beyond algebraic domains and beyond the use of model generators. That is, we can envisage our approach being used for general problems in model generation and constraint solving. While our main motivation remains adding to the mathematical literature on finite algebras, we hope to demonstrate that the notion of specialising problems into sub-problems, then distributing these sub-problems on a Grid architecture has much potential for Artificial Intelligence problem solving.

References

1. S. Colton. *Automated Theory Formation in Pure Mathematics*. Springer Verlag, 2002.

2. S Colton, A Bundy, and T Walsh. On the notion of interestingness in automated mathematical discovery. *International Journal of Human Computer Studies*, 53(3):351–375, 2000.

3. S. Colton, A. Bundy, and T. Walsh. Automatic identification of mathematical concepts. In *Proc. of the 17th International Conference on Machine Learning (ICML2000)*, pages 183–190. Morgan Kaufmann, USA, 2001.

4. S. Colton, A. Meier, V. Sorge, and R. McCasland. Automatic generation of classification theorems for finite algebras. In D. Basin and M. Rusinowitch, editors, *Proc. of the International Joint Conference on Automated Reasoning (IJCAR–2004)*, LNAI. Springer Verlag, 2004. Forthcoming.

5. S. Colton and I. Miguel. Constraint generation via automated theory formation. In *Proc. of CP-01*, 2001.

6. A.M. Frisch, I. Miguel, and T. Walsh. Extensions to proof planning for generating implied constraints. In *Proc. of Calculemus-01*, pages 130–141, 2001.

7. M. Fujita, J. Slaney, and F. Bennett. Automatic Generation of Some Results in Finite Algebra. In R. Bajcsy, editor, *Proc. of the 13th International Joint Conference on Artificial Intelligence (ICJAI)*, pages 52–57. Morgan Kaufmann, USA, 1993.

8. Gap. *GAP Reference Manual*. The GAP Group, School of Mathematical and Computational Sciences, University of St. Andrews, 2000.

9. P Gent, E MacIntyre, P Prosser, and T Walsh. The constrainedness of search. In *Proceedings of AAAI-96*. American Association for Artificial Intelligence, 1996.

10. Carla P. Gomes and Bart Selman. Algorithm portfolios. *Journal of Artificial Intelligence*, 126:43–62, 2001.

11. Carla P. Gomes, Bart Selman, Nuno Crato, and Henry Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24:67–100, 2000.

12. S.R. Kumar, A. Russel, and R. Sundaram. Approximating latin square extensions. *Algorithmica*, 24:128–138, 1999.

13. C. Laywine and G. Mullen, editors. *Discrete Mathematics using Latin Squares*. Wiley, 1998.

14. W.W. McCune. A davis-putnam program and its application to finite first-order model search: quasigroup existence problems. Technical report, Argonne National Laboratory, Division of MSC, 1994.

15. W.W. McCune. *Mace4 Reference Manual and Guide*. Argonne National Laboratory, Division of MSC, 2003. ANL/MCS-TM-264.

16. G L Miller. On the $n^{log_2 n}$ isomorphism technique. Technical Report TR17, The University of Rochester, 1976.

17. J. Slaney. *FINDER, Notes and Guide*. Center for Information Science Research, Australian National University, 1995.

18. J. Slaney, M. Fujita, and M. Stickel. Automated Reasoning and Exhaustive Search: Quasigroup Existence Problems. *Computers and Mathematics with Applications*, 29:115–132, 1995.

19. H. Zhang, M. Bonacina, and H. Hsiang. PSATO: a Distributed Propositional Prover and its Application to Quasigroup Problems. *Journal of Symbolic Computations*, 21:543–560, 1996.

20. H Zhang and J Hsiang. Solving open quasigroup problems by propositional reasoning. In *Proceedings of the International Computer Symposium*, 1994.

21. J. Zhang and H. Zhang. *SEM User's Guide*. Department of Computer Science, University of Iowa, 2001.