Cluster-Based Constraint Ordering for Direct Diagnosis

Muesluem Atas[1] and **Alexander Felfernig**[1] and **Seda Polat Erdeniz**[1] and **Stefan Reiterer**[1] and **Amal Shehadeh**[1] and **Thi Ngoc Trang Tran**[1]

Abstract. Prediction quality and runtime performance are important performance indicators for diagnosis algorithms. In this paper, we propose a new method, which is called CLUSDIAG (*Cluster-Based Constraint Ordered Direct Diagnosis*) which can improve both indicators. CLUSDIAG has a learning phase to find a constraint ordering heuristic. After the learning phase, a diagnosis is found by applying the direct diagnosis algorithm FASTDIAG on an inconsistent constraint set where the constraints are reordered with respect to the constraint ordering heuristic.

Keywords— Configuration Systems; Diagnosis; Constraint Satisfaction; Variable and Value Ordering Heuristics; Clustering; Performance Optimization

1 Introduction

When a constraint set is inconsistent, there is no solution for the corresponding constraint satisfaction problem (CSP) [5]. In this case, a diagnosis is required to make the CSP solvable (s.t. at least one solution can be found). Direct diagnosis algorihms find diagnosis without finding the minimal conflict sets. In diagnosis identification, the most important aspects are the runtime performance of the algorithm and the prediction quality of the result [4].

We propose a new method called CLUSDIAG which can find a diagnoses using cluster-specific [12] constraint ordering heuristics that are exploited by direct diagnosis search (in our case by the FASTDIAG algorithm). These heuristics help to increase the efficiency of diagnosis determination in two different aspects: runtime performance and prediction quality. Time-efficient heuristics find diagnoses faster, whereas prediction quality-efficient heuristics help to find a diagnosis that can be used to provide solutions which are more likely accepted by the user.

Prediction quality-efficient heuristics can be learned based on user interaction logs that contain information on which solution trade-offs were acceptable in the case of inconsistent user requirements. In order to determine the prediction quality of heuristics during the learning phase, we use user interaction data collected within the scope of a user study. Entries in this dataset consist of specified user requirements inconsistent with the knowledge base and corresponding solutions finally selected by the user. In the mentioned user study, we first asked users to specify the customer requirements (or constraints) and then to select alternative solutions if their requirements could not be fulfilled by the underlying configuration knowledge base (represented as a product table).

In our approach we differentiate between two phases: (1) in the offline phase, clustering and learning of cluster-specific heuristics for direct diagnosis (in our case FASTDIAG) is performed, (2) in the online phase, our dataset of user interactions is used to evaluate the performance of FASTDIAG with regard to the aspects of runtime performance and prediction quality.

In order to evaluate the performance of CLUSDIAG, we applied the algorithm on the dataset collected within the scope of our user study. In order to evaluate the prediction quality of the algorithm in combination with a specific constraint ordering heuristic, we analyzed whether a predicted diagnosis leads to a solution selected by the user.

The contributions of our paper are the following. First, the prediction quality of direct diagnosis algorithms can be improved based on our clustering-based learning approach for constraint ordering. Second, our approach can also improve the runtime performance of direct diagnosis search.

The remainder of this paper is organized as follows. In Section 2 we introduce a working example to show the basic CLUSDIAG approach. In Section 3 we report the results of evaluating the approach with regard to runtime performance and prediction quality. The paper is concluded with a discussion of related and future work.

2 Working Example

To show the basic CLUSDIAG approach, we introduce a simple working example. In this example, there are five products available in the product catalog of a shop and each product is further characterized by values related to the three features $f1$, $f2$, and $f3$ as shown in Table 1. We know about six previous customer requirement specifications that lead to an inconsistency, i.e., no solution could be found (see Table 2). Inconsistency in this context means that there does not exist a product in the product table that supports the properties specified by the requirements. Note that for simplicity we assume the existence of equality constraints that specify the relationship between customer requirements and product properties (e.g., $cs1.f1.val = p1.f1.val$). Furthermore, we assume that the set of features describing customer requirements and product properties is equivalent.

Finally, we know which products were selected by the customers after changing their initial requirements in such a way that the new set of requirements is consistent with the product table, i.e., at least one product could be found (see Table 3).

[1] Software Technology Institute, Technical University of Graz, Austria, email {muesluem.atas, alexander.felfernig, amal.shehadeh, spolater, stefan.reiterer, ttrang} @ist.tugraz.at

	p1	p2	p3	p4	p5
f1	1000	1000	600	600	1200
f2	1400	1200	1000	1000	1600
f3	1200	870	1450	720	1100

Table 1. Product table with five products where each has three features.

cust.1 cs1	cust.2 cs2	cust.3 cs3	cust.4 cs4	cust.5 cs5	cus.6 cs6
f1=1400	f1=1000	f1=1400	f1=900	f1=900	f1=500
f2=1000	f2=700	f2=900	f2=700	f2=700	f2=300
f3=1000	f3=700	f3=900	f3=700	f3=700	f3=300

Table 2. Inconsistent constraint sets of six past customers.

2.1 Clustering Phase

In the clustering phase, we cluster similar customer requirements using the former user interaction data (in our working example, this data is represented by Tables 2 and 3). We are now able to apply k-means clustering on the customer requirements.

K-means clustering creates k clusters where it minimizes the sum of squares of distances between cluster elements [9] as shown in Formula 1 where k is the number of target clusters, S is a cluster set, $\mu_{\rm i}$ is the average value of cluster elements in $S_{\rm i}$, and x is a cluster element in $S_{\rm i}$.

$$min \sum_{i=1}^{k} \sum_{x \in S_{\rm i}} \|x - \mu_{\rm i}\|^2 \tag{1}$$

In k-means clustering, the difference (or distance) between two cluster elements x and y with multiple attributes can be calculated based on the *Euclidean Distance (n-dimensional)* [2] as shown in the Formula 2 where $x_{\rm j}$ is the $j^{\rm th}$ attribute of x and $y_{\rm j}$ is the $j^{\rm th}$ attribute of y.

$$x - y = \sqrt{\sum_{j=1}^{n} (x_{\rm j} - y_{\rm j})^2} \tag{2}$$

In our working example, we set the parameters of the k-means clustering algorithm as follows: k=2, i.e., two clusters, maximum number of iterations of k-means clustering: 3. In our working example, k-means clustering is applied to the data mentioned in Table 2.

First, we create the mentioned two clusters and assign the two items with the highest pairwise distance (see Formula 2) as initial cluster elements. The remaining items are now examined in sequence and assigned to the cluster to which they are closest, in terms of Euclidean distance to the cluster mean (or centroid). The mean vector is recalculated each time a new member is added to a cluster and after five steps we established the two clusters as shown in Table 4.

After having created the clusters we cannot yet be sure that each item has been assigned to the right cluster. So, we compare each item's distance to its own cluster mean and to that of the other cluster. Only item $cs2$ is nearer to the mean of the other cluster (cluster2) than to its own cluster (cluster1).

The iterative relocation would now continue up to defined number of maximum iterations. However, in this example each item is now nearer its own cluster mean than to that of the other cluster and the iteration stops after no further adaptations are needed.

cust.1	cust.2	cust.3	cust.4	cust.5	cus.6
p1	p1	p2	p5	p4	p2

Table 3. Products finally selected by customers (specified in Table 2).

	cluster1		cluster2	
	items	centroid	items	centroid
s.1	cs1	[1400,1000,1000]	cs6	[500,300,500]
s.2	cs1,cs2	[1200,850,850]	cs6	[500,300,500]
s.3	cs1,cs2,cs3	[1266,866,866]	cs6	[500,300,500]
s.4	cs1,cs2,cs3	[1266,866,866]	cs6,cs4	[700,500,500]
s.5	cs1,cs2,cs3	[1266,866,866]	cs6,cs4,cs5	[766,566,566]

Table 4. Clustering customer requirements in five steps.

	cluster1		cluster2	
	items	centroid	items	centroid
i.1	cs1,cs2,cs3	[1266,866,866]	cs6,cs4,cs5	[766,566,566]
i.2	cs1,cs3	[1400,950,950]	cs6,cs4,cs5,cs2	[825,600,600]

Table 5. Iterations of relocating constraint sets.

2.2 Learning Phase

After clustering the user requirements specified in the user interaction log (see Table 2), we apply a genetic algorithm for learning two different constraint ordering heuristics: one for optimizing the prediction quality of direct diagnosis and the other one for optimizing the runtime performance per cluster.[2]

In the working example, the genetic algorithm learns constraint ordering heuristics for improving the prediction quality. Prediction quality in our case is measured in terms of precision ([8]), i.e., the overall share of correct diagnosis predictions (diagnoses that lead to a product that also has been selected by the user) in relation to the total number of diagnosis predictions (determined by FastDiag on the basis of the constraint ordering within a cluster). We initialize the genetic algorithm with the parameters population size = 2, maximum number of generations = 2, mutation rate = 0.015, uniform rate = 0.5, and target fitness value = 1.

In the first generation, for cluster-1, we create two (since population size is set to two) random constraint orderings: [c2,c1,c3], [c3,c2,c1]. Then one by one we use these constraint orderings as constraint orderings in the direct diagnosis algorithm FastDiag [6]. The prediction quality of the [c2,c1,c3] for cs1 is calculated 0.5, which means the average of the diagnosis algorithm's prediction qualities of cluster elements (where the constraint ordering heuristic [c2,c1,c3] is applied) is 0.5.

We calculate the precision values for all constraint orderings in the populations of each generation and select the best individuals (individual is constraint ordering in our case) in the populations.

	cluster1		cluster2	
	ordering	fitness	ordering	fitness
gen.0	[c2,c3,c1]	0.4	[c3,c2,c1]	0.3
gen.1	[c3,c2,c1]	0.5	[c3,c2,c1]	0.3
gen.2	[c3,c2,c1]	0.5	[c1,c3,c2]	0.6

Table 6. Learning constraint ordering heuristic for each cluster.

After 2 generations (maximum number of generations is set to 2), the genetic algorithm stops. Then we can use the best constraint ordering learned by the genetic algorithm. As shown in Table 6, the best constraint ordering for cluster1 is [c3,c2,c1] whereas [c1,c3,c2] for cluster2.

[2] Combined heuristics allowing tradeoffs between runtime performance and prediction quality will be analyzed within the scope of our future work.

We apply the same algorithm for learning constraint ordering heuristics for runtime performance. In this case we set the target fitness value to 0 because this time the fitness value is the runtime which we aim to minimize this time.

2.3 Application Phase

In the offline phase, we focus on cluster generation and learning of heuristics for direct diagnosis. Now, in the online phase, we apply our heuristics to a new inconsistent set of customer requirements.

	c1	c2	c3
cs_new	f1=1200	f2=1000	f3=1000

Table 7. A new set of customer requirements (represented in terms of constraints).

In the working example, we need to find a diagnosis with a high prediction quality since our aim is to find a product of high relevance for the customer.

First, we find a cluster that is closest to the new set of customer requirements cs_new[1200,1000,1000]. This is cluster1 with centroid [1400,950,950]. Then, we apply the identified optimal constraint ordering for cluster1 (which is [c3,c2,c1]) to cs_new. The reordered constraint set is cs_new_reordered = {(f3=1000), (f2=1000), (f1=1200)}.

When we apply the FastDiag algorithm to cs_new_reordered, we find the diagnosis {c1,c3}. After removing the diagnosis from the constraint set, we have cs_new_reordered_diagnosed = {(f2=1000)}. According to this new constraint set, there are two solutions available in the product table which are p3 and p4.

After applying the diagnosis by removing them from the customer's inconsistent constraint set, we can find two products which can be recommended.

3 Evaluation

In order to evaluate ClusDiag, we applied our approach to a dataset collected within the scope of a user study - the dataset comprises 264 different inconsistent requirements specifications and the corresponding products users (study participants) selected after changing their requirements. In the mentioned user study, the product table consisted of 30 digital cameras which were described by 10 features.

We compared the mentioned two versions of ClusDiag with each other and with the FastDiag algorithm with randomized constraint ordering (baseline version). Figure 1 shows a comparison of ClusDiag-enhanced FastDiag (FastDiag that exploits the cluster-based constraint ordering heuristics) and a basic FastDiag version (constraints are ordered randomly) with regard to *prediction quality.* In Figure 2, we show the comparison of the runtime performance of FastDiag with the corresponding clustering-supported version of the algorithm (in this case, heuristics were optimized with regard to algorithm runtime performance).

4 Related Work

Most widely known algorithms for the identification of minimal diagnoses are QuickXplain [10] for predetermining minimal conflict sets which are used by a hitting set based approach such as HSDAG [13] to compute minimal diagnoses. FastDiag [6] also determines minimal diagnoses but does this without the need of identifying minimal conflict sets.

De Kleer et al. [3] introduce a probability-based approach to estimate relevant diagnoses. This approach is based on traditional hitting set based diagnosis determination, i.e., is in the need of predetermining conflict sets.

Methods to find approximate solutions for diagnosis tasks are introduced, for example, in [1, 14]. In contrast to our approach, approximate solutions do not guarantee the minimality of diagnoses. Since the initial version of HSDAG introduced by [12], a couple of algorithms have been proposed to increase the efficiency of the inital version – see, for example, [15].

The authors of [11] focus on interactive settings where users of constraint-based applications are confronted with situations where no solution can be found. In this context, the authors propose an approach to determine representative sets of diagnoses, i.e., diagnoses that cover as much as possible all potential faulty elements of the solution space. Direct diagnosis as used in our work focuses on constraint orderings that help to increase the probability of finding diagnoses relevant for the user. Focusing on such leading diagnosis is not the central focus of the work presented in [11].

Authors of [7] present an algorithm for computing a generalization of conflict-based explanations of inconsistency for the QCSP (Quantified Constraint Satisfaction Problem) which is a generalization of the classical CSP in which some of the variables can be universally quantified.

Direct diagnosis is an approach to omit conflict detection and directly determine diagnoses without the need to identify the corresponding conflict sets. FastDiag [6] is a diagnosis algorithm that supports direct diagnosis. It is based on a divide-and-conquer strategy with a number of consistency checks similar to QuickXplain [10]. QuickXplain [10] itself is a conflict detection algorithm that helps to identify minimal conflict sets. Similar to FastDiag, the algorithm relies on a linear ordering of the constraints. It is often used in combination with HSDAG to determine diagnoses [13].

5 Conclusions

In this paper, we have introduced a new diagnosis approach called ClusDiag that increases the efficiency of direct diagnosis algorithms with respect to prediction quality and runtime by applying cluster-specific constraint ordering heuristics.

As future work, we will test our approach with different clustering techniques such as hierarchical clustering. Furthermore we will include further optimization criteria for learning constraint ordering heuristics. For example, we will learn constraint ordering heuristics that support trade-offs between runtime performance and prediction quality.

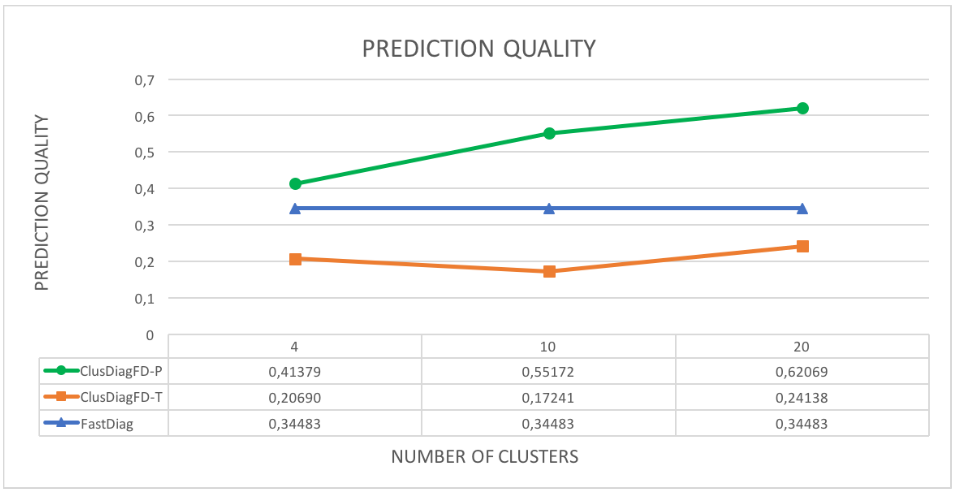
Acknowledgement

The work presented in this paper has been conducted within the scope of the EU Horizon 2020 project AGILE (http://agile-iot.eu/).

REFERENCES

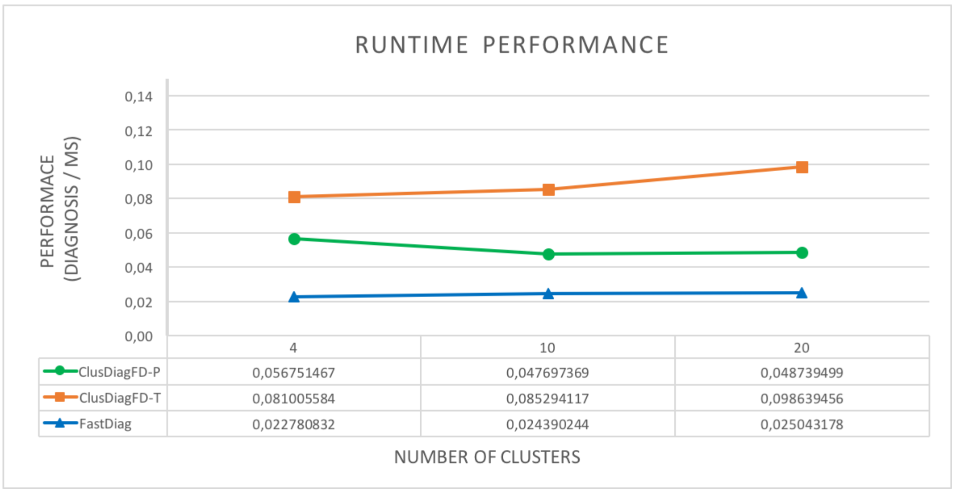
[1] Rui Abreu and Arjan JC Van Gemund, 'A low-cost approximate minimal hitting set algorithm and its application to model-based diagnosis.', in *SARA*, volume 9, pp. 2–9, (2009).

[2] Per-Erik Danielsson, 'Euclidean distance mapping', *Computer Graphics and image processing*, **14**(3), 227–248, (1980).



	4	10	20
ClusDiagFD-P	0,41379	0,55172	0,62069
ClusDiagFD-T	0,20690	0,17241	0,24138
FastDiag	0,34483	0,34483	0,34483

Figure 1. Comparison of the prediction quality (measured in terms of precision [0..1]) of the different variants of FastDiag (ClusDiagFD-P = FastDiag with constraint orderings optimized for high prediction quality, ClusDiagFD-T = FastDiag with heuristics focusing on runtime performance, FastDiag = basic version with randomized constraint orderings). We ran the algorithms over 264 inconsistent CSPs (dataset collected within the scope of a user study). We observed that increasing the number of clusters also helps to increase the prediction quality.



	4	10	20
ClusDiagFD-P	0,056751467	0,047697369	0,048739499
ClusDiagFD-T	0,081005584	0,085294117	0,098639456
FastDiag	0,022780832	0,024390244	0,025043178

Figure 2. Runtime performance comparison of diagnosis algorithms. We ran the algorithms over 264 inconsistent CSPs (from user study data). Then we have taken the average of the runtimes (over 264 CSPs) needed for diagnosis identification. Performance denotes the number of diagnoses found per millisecond. When we increase the number of clusters, diagnosis speed is increasing which means performance of ClusDiag increases.

[3] Johan de Kleer, 'Using crude probability estimates to guide diagnosis', *Artificial Intelligence*, **45**(3), 381–391, (1990).

[4] Alexander Felfernig, Gerhard Friedrich, Monika Schubert, Monika Mandl, Markus Mairitsch, and Erich Teppan, 'Plausible repairs for inconsistent requirements.', in *IJCAI*, volume 9, pp. 791–796, (2009).

[5] Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen, *Knowledge-based Configuration: From Research to Business Cases*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1 edn., 2014.

[6] Alexander Felfernig, Monika Schubert, and Christoph Zehentner, 'An efficient diagnosis algorithm for inconsistent constraint sets', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **26**(01), 53–62, (2012).

[7] Alex Ferguson and Barry O'Sullivan, 'Quantified constraint satisfaction problems: From relaxations to explanations.', in *IJCAI*, pp. 74–79, (2007).

[8] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich, *Recommender systems: an introduction*, Cambridge University Press, 2010.

[9] Xin Jin and Jiawei Han, *K-Means Clustering*, 563–564, Springer US, Boston, MA, 2010.

[10] Ulrich Junker, 'Quickxplain: Conflict detection for arbitrary constraint propagation algorithms', in *IJCAI'01 Workshop on Modelling and Solving problems with constraints*, (2001).

[11] Barry O'Sullivan, Alexandre Papadopoulos, Boi Faltings, and Pearl Pu, 'Representative explanations for over-constrained problems', in *AAAI*, volume 7, pp. 323–328, (2007).

[12] Seda Polat Erdeniz, Alexander Felfernig, and Muesluem Atas, 'Cluster-specific heuristics for constraint solving', *International Conference on Industrial, Engineering, Other Applications of Applied Intelligent Systems*, (2017).

[13] Raymond Reiter, 'A theory of diagnosis from first principles', *Artificial intelligence*, **32**(1), 57–95, (1987).

[14] Staal Vinterbo and Aleksander Øhrn, 'Minimal approximate hitting sets and rule templates', *International Journal of approximate reasoning*, **25**(2), 123–143, (2000).

[15] Franz Wotawa, 'A variant of reiter's hitting-set algorithm', *Information Processing Letters*, **79**(1), 45–51, (2001).