Engineering Optimization
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/geno20

Mining constraint relationships and redundancies with association analysis for optimization problem formulation

Adam Cutbill[a] & G. Gary Wang[a]
[a] School of Mechatronic Systems Engineering, Simon Fraser University, Product Design and Optimization Laboratory, Surrey, Canada
Published online: 06 Jan 2015.

Engineering Optimization, 2015
http://dx.doi.org/10.1080/0305215X.2014.995177

Taylor & Francis
Taylor & Francis Group

Mining constraint relationships and redundancies with association analysis for optimization problem formulation

Adam Cutbill and G. Gary Wang*

School of Mechatronic Systems Engineering, Simon Fraser University, Product Design and Optimization Laboratory, Surrey, Canada

(*Received 4 August 2014; accepted 1 December 2014*)

Constraints are necessary in optimization problems to steer optimization algorithms away from solutions which are not feasible or practical. However, redundant constraints are often added, which needlessly complicate the problem's description. This article introduces a probabilistic method to identify redundant inequality constraints for black-box optimization problems. The method uses Jaccard similarity to find item groups where the occurrence of a single item implies the occurrence of all other items in the group. The remaining groups are then mined with association analysis. Furthermore, unnecessary constraints are classified as redundant owing to co-occurrence, implication or covering. These classifications are presented as rules (in readable text), to indicate the relationships among constraints. The algorithm is applied to mathematical problems and to the engineering design of a pressure vessel. It was found that the rules are informative and correct, based on the available samples. Limitations of the proposed methods are also discussed.

Keywords: constraint handling; optimization; data mining; association analysis; constraint redundancy; constraint relationships; black-box optimization

1. Introduction

In optimization, constraints are often defined to eliminate solutions which are not physically feasible or practical. Unfortunately, in some cases, designers may unknowingly or inadvertently specify constraints which are redundant. Common causes and consequences of superfluous constraints have been discussed by Karwan *et al.* (1983). In particular, they complicate the problem formulation, may represent a mistake in the problem formulation, and may impact the performance of the optimization method. Although redundant constraint removal has been explored in depth for mathematical programming (Karwan *et al.* 1983), there is little work related to identifying redundant black-box constraints, which have no algebraic expression. In this article, a probabilistic method, including *a priori* association analysis, is proposed to identify and present constraint redundancies as rules.

Association analysis is a popular data-mining technique, introduced in the 1990s, for finding implication rules in Boolean datasets (Agrawal, Imielinksi, and Swami 1993a, 1993b). The method has been used extensively for market basket analysis to find items which affect sales of other items (*e.g.* 70% of people who buy bread also buy butter). Items are grouped into sets

*Corresponding author. Email: gary_wang@sfu.ca

(itemsets) from which implication rules are mined. Optimization constraints can also be treated as Boolean variables: a 1 indicates a violation and a 0 indicates satisfaction. In this context, a rule $A \rightarrow B$ means that A is redundant due to B in P percentage of cases.

However, association rules are only one of the useful patterns that may be mined from Boolean variables. In some cases, it may be more informative to provide rules in the form "If any ($\boldsymbol{A}$), then all ($\boldsymbol{A}$)", where $\boldsymbol{A}$ is a group of items. In market basket mining, this pattern infers that A is sold as a bundle. In constraint mining, this pattern infers that constraints co-occur, and that only one constraint in the set is required. Some work has been done to combine implication rules into bundles. The hyperclique pattern gives a minimum probability that items co-occur using a metric called *h-conf* (Xiong, Tan, and Kumar 2006). The method presented by Huang *et al*. (2006) involves merging association rules *a posteriori*, instead of identifying co-occurring items beforehand.

This article first presents past approaches and definitions related to redundant constraint identification. Next, a method is introduced to efficiently test for co-occurring item groups using Jaccard similarity, before mining the remaining itemsets with the *a priori* association analysis method. These steps identify relationships between constraints. Subsequently, constraints which do not co-occur and are not implicit are evaluated to see whether they uniquely restrict the feasible space. The method is also applied to find redundant constraints in an engineering design context, which is a new application of association analysis that benefits from the proposed technique.

2. Background

2.1. *Constraints in black-box engineering design*

In engineering design, inequality constraints are typically written as a vector of functions as in Equation (1):

$$\boldsymbol{g}(\boldsymbol{x}) \leq 0 \tag{1}$$

In practice, $\boldsymbol{g}(\boldsymbol{x})$ may contain nonlinear functions or even complex engineering simulations, such as finite element analysis or computational fluid dynamics. For such cases, it may be impossible to identify redundant constraints algebraically as done in previous work (Thompson, Tonge, and Zionts 1966; Karwan *et al*. 1983; Caron, McDonald, and Ponic 1989; Paulraj and Sumathi 2010); no assumptions can be made about the properties of the functions in $\boldsymbol{g}$ (*e.g.* linearity, convexity, continuity). Instead, the constraints can be thought of as black boxes, where only the input and output are considered. In this situation, random test points within the optimization bounds may be sampled, with 0s indicating a constraint pass and 1s representing a constraint violation. From this basic information, constraints may be analysed as a Boolean dataset.

2.2. *Constraint redundancy identification methods*

Redundant constraint identification has been the subject of extensive research since Boot's (1962) pioneering work. Although progress has been made, most work has focused on classifying and reducing constraints for linear programming. For example, Brearley, Mitra, and Williams (1975) test design variables at their lower and upper bounds, and use the constraints' coefficients to see whether it is possible to satisfy them within the chosen range. Telgen and co-workers developed a deterministic approach, similar to the simplex method in linear programming, using a minimum

ratio test and simplex tableaux (Karwan *et al*. 1983). Unfortunately, the majority of deterministic constraint reduction techniques require linear algebraic functions. A comparative study of mathematical programming methods was performed by Paulraj and Sumathi (2010).

Hit-and-run methods form a radically different approach, using sampling to identify redundancy. These methods are not limited to linear problems (Berbee *et al*. 1987). The trade-off is that hit-and-run does not guarantee correctness. Functionally, these methods extend rays from chosen feasible points, and determine where the rays collide with the surrounding constraints. If a constraint is hit, it is necessary and non-redundant. Unfortunately, hit-and-run requires the constraints to form a bounded, non-empty feasible space. In addition, to detect collisions between the rays and constraints, an approximation of the constraint's surface is required. These requirements are not met by black-box problems, where feasible regions may be discontinuous, highly nonlinear or expensive to approximate.

The set-covering (SC) approach (Boneh 1984; Feng 1999) is the most general method, and the most suitable for black-box problems. Set covering represents the feasibility of constraints for a given sample point $\boldsymbol{x}^i$, as a binary vector $\boldsymbol{e}_i \in \boldsymbol{E}$ (called an observation). If a constraint $g_j$is violated at the point $\boldsymbol{x}^i$, $e_{ij} \epsilon \boldsymbol{e}_i$ is 1. Otherwise, if the constraint is feasible, e_{ij} is 0. The main theorem of the SC approach states that if any sample point violates one or more constraints $\left(\sum \boldsymbol{e}_i \geq 1\right)$, then at least one of those constraints is necessary. Furthermore, a vector $\boldsymbol{Y}$ may be defined to summarize which constraints are necessary or redundant, based on the given data. Specifically, $y_j \epsilon \boldsymbol{Y}$ is 1 if constraint g_j is necessary, and y_j is 0 if g_j is redundant. With this notation, Boneh (1984) represents the main theorem of the SC approach as in Equation (2):

$$\boldsymbol{EY} \geq (1, 1, 1 \ldots)^T \tag{2}$$

Given a sample of observations, Equation (2) can be solved for $\boldsymbol{Y}$ to identify redundant constraints. Of course, the solutions of $\boldsymbol{Y}$ are not necessarily unique. Therefore, Boneh (1984) suggests pursuing the solution which minimizes computational cost, and presents an algorithm to do so: the SC algorithm. The method in this article differs from SC in that its main concern is not finding irreducible sets of constraints (solutions of $\boldsymbol{Y}$). Instead, the proposed method returns information regarding *why* constraints are redundant. Specifically, the article introduces methods to determine whether constraints co-occur, are implicit with respect to another constraint or are covered by other constraints. These relationships are formalized in this article.

2.3. *Constraint redundancy as rules*

According to a survey by Karwan *et al*. (1983), the general consensus is that a constraint is redundant if its removal has no effect on the feasible space. A more specific taxonomy of redundancy is also presented in the survey (Karwan *et al*. 1983). For example, an inequality constraint is *strongly redundant* if the region it eliminates does not intersect with the feasible region. Similarly, a *weakly redundant* inequality only intersects the feasible region at the boundary. In practice, relationships between constraints can be helpful for a designer who is trying to find errors in their problem formulation. Therefore, the following definitions are proposed not only to indicate *if* a constraint is redundant, but also to indicate *why*.

2.3.1. *Redundant due to co-occurrence*

Conceptually, co-occurrence can be thought of as a set of constraints ($\boldsymbol{C_B}$) being violated as a bundle. Imagine the sale of items such that you cannot buy one without the other; the purchase of any item implies the sale of the others. It means that although the constraints or simulations may be conceptually different, they have the same effect on the feasible region. This implies

Table 1. Implication example.

P	Q
1	1
1	0
1	1
0	0

that any item C_k from $\boldsymbol{C}_B$ can be chosen as a representative, while the remaining constraints are suppressed. Co-occurrence is considered to be the most informative type of redundancy. If constraints *always* co-occur they are *duplicates*. In this article, a relaxed definition of co-occurrence is given in Section 3.1 based on Jaccard similarity (*i.e.* if constraints are nearly duplicates, but not exactly, a rule may still be generated).

DEFINITION 2.1 (REDUNDANT DUE TO CO-OCCURRENCE): *Let $\boldsymbol{C_B}$ be a set of inequality constraints. If $\boldsymbol{C}_B$ co-occurs, then a constraint $C_k \in \boldsymbol{C}_B$ can be chosen as the representative of $\boldsymbol{C_B}$, and the remaining constraints $\boldsymbol{C_B}\backslash\{C_k\}$ are redundant.*

2.3.2. *Redundant due to implication*

Redundancy due to implication means that a constraint is dominated by another constraint, within a given probability. Consider the data in Table 1. In the rows where Q has a value of 1, P also has a value of 1, but not *vice versa*. Therefore, we can write $Q \rightarrow P$. Another way to think of this is that Q is dominated by P, or if there is Q, then there is P. If these items are constraints and a 1 means a violation, Q is redundant due to P by implication. This rule is considered to be the second most informative type of rule. Association rules are mined efficiently using *a priori* association analysis, as explained in Section 3.2.

DEFINITION 2.2 (REDUNDANT DUE TO IMPLICATION)*: Let C_i and C_j be non-co-occurring inequality constraints. If $C_i \rightarrow C_j$, then C_i is redundant due to C_j by implication.*

2.3.3. *Redundant due to covering*

Redundancy due to covering is equivalent to the general definition of redundancy in SC (Feng 1999). Namely, if a constraint C_k can be removed without affecting the feasible region, it is redundant. Note that if each time there is a violation in C_k, at least one other constraint is also violated, C_k does not have a unique effect on the feasible region. In this article, C_k is called redundant due to covering (*i.e.* C_k is *covered* by other constraints). In addition, covering is only evaluated on the list of non-co-occurring and non-implicit constraints. It is argued that co-occurrence and implicitness is more informative to describe redundancies, as these rules include which constraint(s) make a constraint redundant. The constraints which cover C_k, however, are not recorded in this article for simplicity. Consequently, covering is the least informative rule.

DEFINITION 2.3 (REDUNDANT DUE TO COVERING)*: Let $\boldsymbol{C_T}$ be the set of all constraints which are non-co-occurring and non-implicit. If no infeasible observation is made feasible by the removal of a constraint $C_k \in \boldsymbol{C_T}$, C_k is redundant.*

3. The constraint-mining method

An overview of the sequence for mining constraints is described in Figure 1. Each step is described in detail in the following sections. The first four steps are general, and can be applied to rule mining for any Boolean dataset. The remaining steps are specific to the application of constraint mining.

3.1. *Identifying frequent itemsets and co-occurring items*

Although the outputs of constraint functions are not usually Boolean, they can be treated as such (as in SC). In this section, the terminology from data mining will be employed to emphasize that Steps 1–4 are not restricted to constraint mining. An *item* indicates a generic attribute/column (*e.g.* a constraint), while an *observation* indicates a particular sample/row (*e.g.* a vector of design variable values). An item *occurs* in an observation if its value is 1 (*e.g.* a violation for the sample). A group of items is an *itemset*. An example is given in Table 2 for five observations (samples) and six items (constraints).

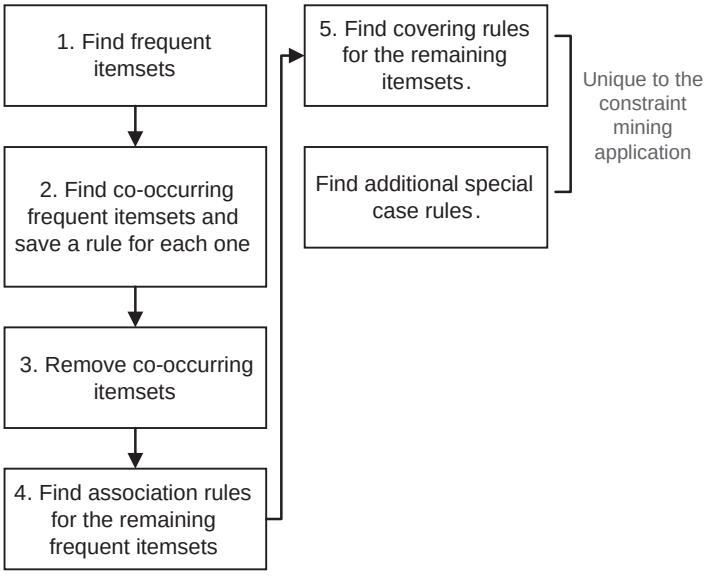


Figure 1. Overview of the constraint-mining method.

Table 2. Example binary representation of constraint violations.

Observation	$C1$	$C2$	$C3$	$C4$	$C5$	$C6$
1	1	1	1	0	1	1
2	1	1	1	0	0	1
3	0	0	1	0	1	1
4	1	0	0	0	0	0
5	0	1	0	1	0	0

Note: 1 = violation; 0 = no violation.

3.1.1. *Step 1: Filter items by support and find frequent itemsets*

As explained by Tan, the number of possible association rules from a brute-force analysis of d number of items is $\text{nRules} = 3^d - 2^{d+1} + 1$ (Tan, Steinbach, and Kumar 2006). For example, in Table 2 d is 6, resulting in 602 possible rules. This makes testing every combination of items computationally intensive. Therefore, itemsets which have few occurrences are eliminated in the general *a priori* association analysis method (Agrawal, Imielinksi, and Swami 1993a, 1993b). The frequencies of itemsets are quantified by Support s, as in Equation (3):

$$s(I) : \frac{\sigma(\boldsymbol{I})}{N} \tag{3}$$

where $\sigma(\boldsymbol{I})$ is the count of observations such that *all* items in itemset $\boldsymbol{I}$ occur, and N is the total number of observations. For example, from Table 2, the support of $\{C1, C2\}$ together is $s(C1, C2) = (2/5)$. An itemset is considered *frequent* if $s(\boldsymbol{I}) \geq$ minsup, where minsup is a parameter chosen by the user. A lower minsup means that rarer items will be mined for rules, but will also increase the computational effort. Although filtering using frequency is important for transactional databases with thousands of items, in most engineering problems the number of constraints is much lower, as constraints are defined based on physical properties. Therefore, minsup can be set to 0 or a small number for this application. In Section 3.5.1, it is explained that support is also informative to indicate which constraints occur rarely (if ever).

In the *a priori* algorithm, frequent itemsets are built iteratively, from the bottom up, by combining only the frequent itemsets found *a priori*. As support is anti-monotonic, only the frequent k-sized sets are needed to generate the frequent $k + 1$-sized sets. Support will only decrease by adding new items. The flowchart in Figure 2 lays out the generation phase of *a priori* association analysis. Efficient implementation details can be found in most introductory data-mining books.

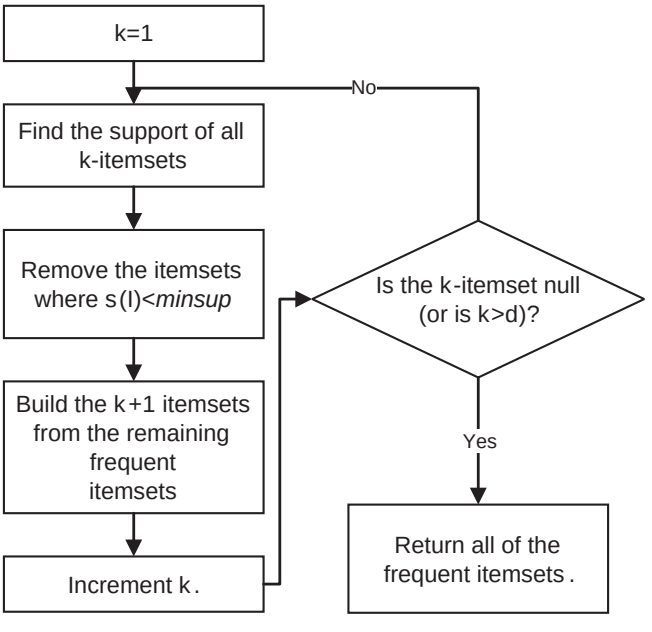


Figure 2. Generation of a frequent itemset of size k.

3.1.2. *Step 2: Use the Jaccard measure to identify co-occurring sets*

Once frequent itemsets have been generated, they can be checked for co-occurrence. This stage is a contribution of this article, and not part of Agrawal's standard *a priori* method. The Jaccard index j is a symmetric measure that represents the conditional probability of co-occurrence if it is known that one item occurs: $j(a1, a2) = P(\ a1 \wedge a2\ \mid a1 \vee a2)$. Jaccard for the frequent itemset $\boldsymbol{I}$ can be written as in Equation (4):

$$j(I) = \frac{\sigma(\boldsymbol{I})}{\gamma(\boldsymbol{I})} \tag{4}$$

where γ is the count of observations such that *any* item in $\boldsymbol{I}$ occurs. Sequentially, γ is evaluated and stored during the computations of $s(\boldsymbol{I})$ for frequent itemsets (as an additional step). In the proposed method, if $j(\boldsymbol{I})$ is greater than a co-occurrence threshold minjacc, the itemset is called co-occurring or bundled. Although the bundle pattern has been discussed before by Huang *et al.* (2006), it was not defined using Jaccard. The mathematical meaning of *bundle* in this context is different from that of Huang's, but is conceptually similar in that it indicates co-occurrence.

DEFINITION 3.1 (CO-OCCURRENCE/BUNDLING): *An itemset* $\boldsymbol{I}$ *co-occurs (is bundled) if* $j(\boldsymbol{I}) \geq$ minjacc.

It is important to emphasize that $j(\boldsymbol{I})$ is fundamentally different from $s(\boldsymbol{I})$. Support is a measure of the itemset's likelihood to *occur*, not of the *co-occurrence* of items in the set. For example, consider the data in Table 2: $s(C3, C6)$ is 0.6. On the other hand, $j(C3, C6)$ is 1. Jaccard's denominator only counts the observations where either $C3$ or $C6$ occurs in the denominator. If $j = 1$ for an itemset, the items are duplicates. Therefore, by setting minjacc to 1, the exact duplicate item groups will be found. By reducing minjacc, items that are nearly duplicates will be found.

3.1.3. *Step 3: Remove co-occurring frequent itemsets*

If a set of items is co-occurring, a rule is saved for those items (*e.g.* '*Constraints C3 and C6 co-occur*"'). In addition, if minjacc is chosen to exceed the threshold for association rules (minconf), there is no need to test association rules on itemsets that co-occur. This is explained in Section 3.2.1. Furthermore, it can be shown that the subsets of co-occurring sets also co-occur, based on the fact that j is anti-monotonic with the addition of items.

THEOREM 3.1 *Given a co-occurring itemset* $\boldsymbol{C_B}$ *and a subset* $\boldsymbol{C_B^-} \subseteq \boldsymbol{C_B}$*:* $j(\boldsymbol{C_B^-}) \geq j(\boldsymbol{C_B})$ *. Therefore, if* $\boldsymbol{C_B}$ *is co-occurring,* $\boldsymbol{C_B^-}$ *is co-occurring.*

Proof

$$\sigma(\boldsymbol{C_B}) \leq \sigma(\boldsymbol{C_B^-}) \text{ by the anti} - \text{monotonic property of support.}$$

$$\gamma(\boldsymbol{C_B^-}) \leq \gamma(\boldsymbol{C_B}) \text{ by the definitions } \gamma(\boldsymbol{C_B}) = |\text{any}(\boldsymbol{C_B})| \text{ and } \boldsymbol{C_B^-} \subseteq \boldsymbol{C_B}.$$

$$\therefore \text{minjacc} \leq \frac{\sigma(\boldsymbol{C_B})}{\gamma(\boldsymbol{C_B})} \leq \frac{\sigma(\boldsymbol{C_B^-})}{\gamma(\boldsymbol{C_B^-})}$$

■

Theorem 3.1 means the subsets of a bundle will not generate additional interesting rules. For instance, if a rule states that items $\{a, b, c\}$ co-occur with high probability, no new information is

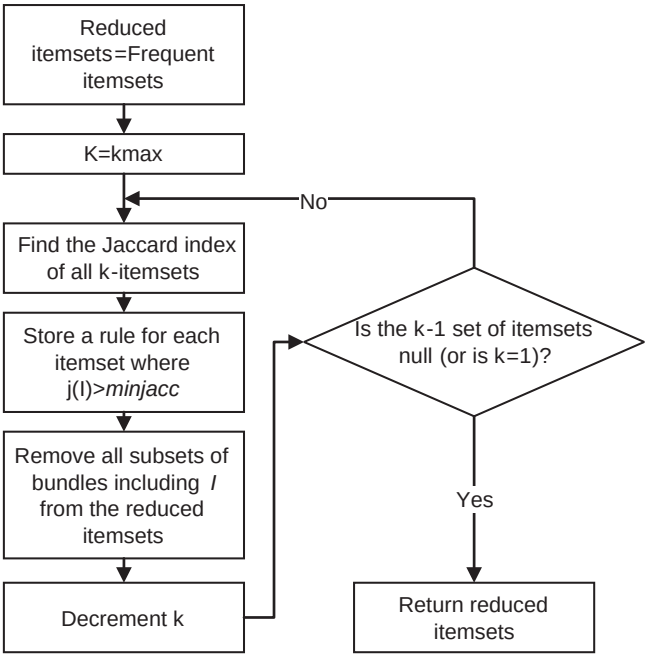


Figure 3. Identification of bundles and itemset reduction.

gained by testing the sets $\{a, b\}$,$\{b, c\}$ or $\{a, c\}$. Therefore, subsets of item bundles can be omitted from future rule mining.

To remove bundle subsets efficiently, the following algorithm can be used. Starting with $k_{\max}$, $j(\boldsymbol{I})$ is found for each frequent itemset of size k. Each time $\boldsymbol{I}$ is co-occurring, a rule is saved, and all of its subsets (including $\boldsymbol{I}$ itself) are removed. Once each set of size k has been evaluated, k is decremented. The process stops if all sets of size k co-occur, or if $k = 1$. The flowchart in Figure 3 describes the process of bundle identification and itemset reduction. Note that the removal of co-occurring itemsets is performed on a copy of the original list. The original list is kept, as σ values from the removed itemsets may be necessary to mine associations on the non-co-occurring itemsets.

3.2. *Identifying association rules*

Association rule mining is a well-researched area in data mining popularized by Agrawal (Agrawal, Imielinksi, and Swami 1993a, 1993b). Since its introduction, a significant amount of research has been published that presents efficient algorithms (Han *et al*. 2004; Zhou and Yau 2007), new patterns (Özden, Ramaswamy, and Sillberschatz 1998; Cheng, Yu, and Han 2006; Xiong, Tan, and Kumar 2006), compact representation of itemsets (Pasquier *et al*. 1999) and new measures of rule interestingness (Tan, Kumar, and Srivastava 2002). Tan and colleagues provide an excellent review of the advances in the field in the bibliographical notes of their book (Tan, Steinbach, and Kumar 2006). Applications have also expanded beyond market baskets to finding protein interactions (Atluri *et al*. 2009) and associations between carbon levels and ocean climates (Potter *et al*. 2003).

The rule-generation and confidence-pruning step of Agrawal's *a priori* algorithm is described in this section. Rule generation and confidence pruning usually directly follow support filtering. However, this is pushed to a fourth step with the additions of bundle identification and removal.

This stage is also only performed on the itemsets that are not bundles. This greatly reduces the number of implications rules mined unnecessarily, as explained in Section 3.2.2.

3.2.1. *Step 4: Generate association rules*

Association rules are generated based on a measured called *confidence* in Agrawal's algorithm (Agrawal, Imielinksi, and Swami 1993a, 1993b). Confidence c represents the conditional probability of $\boldsymbol{C}$ given $\boldsymbol{A}$, based on the data available. It is defined by Equation (5), where $\boldsymbol{A}$ and $\boldsymbol{C}$ are frequent itemsets:

$$c(\boldsymbol{A} \rightarrow \boldsymbol{C}) : \frac{\sigma(\boldsymbol{A} \cup \boldsymbol{C})}{\sigma(\boldsymbol{A})} \tag{5}$$

In the association rule $\boldsymbol{A} \rightarrow \boldsymbol{C}$, $\boldsymbol{A}$ is the *antecedent* set and $\boldsymbol{C}$ is the *consequent* set. Rules are generated by testing confidence against a threshold minconf. The appropriate choice for minconf, like minsup or minjacc, also depends on the application. For example, in design engineering, it is important that rules are correct to avoid suppressing constraints which are not truly redundant. Therefore, as a rule of thumb for constraint mining, minconf should be nearly or exactly 1, as explained in Section 4.2.

Furthermore, minconf must also be equal to or less than minjacc. Otherwise, a case may exist where the items will be classified as co-occurring, when there is a higher probability of implication. For instance, consider the set $\{C3, C5\}$, from Table 2: $j\{C3, C5\} = 0.67$, $c\{C5 \rightarrow C3\} = 0.67$ and $c\{C3 \rightarrow C5\} = 1$. If minjacc were set lower than 0.67, the association $c\{C3 \rightarrow C5\}$ would have been missed. By enforcing that minjacc $\geq$ minconf, by definition, a co-occurrence rule always has equal or greater probability than the association rules for the same itemset. Thus, bundles (and their subsets) can be safely removed before the association-mining stage.

Association rules are identified by building up m-sized consequents from the consequents which formed confident rules *a priori*. As an example, consider the set $\boldsymbol{I} = \{a, b, c, d\}$. Assume

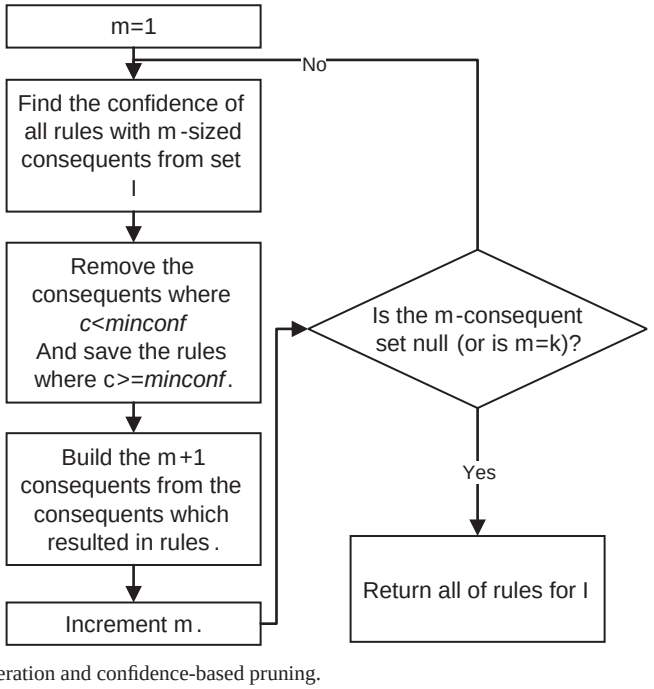


Figure 4. Rule generation and confidence-based pruning.

that rules $\{a, b, c\} \rightarrow \{d\}$ and $\{b, c, d\} \rightarrow \{a\}$ are the single-item consequent rules that pass minconf. In the next iteration, $\{a, d\}$ is tested as a two-item consequent (Rule : $\{b, c\} \rightarrow \{a, d\}$). Figure 4 shows the rule-generation process for a frequent itemset using confidence-based pruning, which is also described in detail by Tan, Steinbach, and Kumar (2006). This process is repeated on each frequent itemset.

3.2.2. *A brief discussion regarding the number of rules and the benefit of co-occurring sets*

Although the support-confidence framework reduces the number of possible implications between items, a dataset with similar high-support items creates many rules. This is a fundamental drawback of the standard *a priori* algorithm. A large set of implication rules is difficult for users to interpret or for a pruning algorithm to reduce.

For example, if items A, B and C are all frequent and likely to co-occur, the frequent itemsets may be $\{A\}, \{B\}, \{C\}, \{AB\}, \{AC\}, \{BC\}, \{ABC\}$. In total, there may be 18 possible association rules to decipher for these three items. Meanwhile, a single rule stating that '$\{A, B, C\}$ co-occur' conveys the same information. In fact, this rule is not only more compact, but also simpler to understand and take action. This is why co-occurring sets are considered more informative than associations.

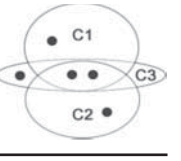
3.3. *Further rule reductions when considering item removal*

3.3.1. *Restricting association rules to one-item antecedents*

This section shows that association rules are only needed for one-item antecedents if the task of concern is item removal (*e.g.* suppressing constraints). In general, high confidence rules of the form $\boldsymbol{A} \rightarrow \boldsymbol{C}$ are *not* implied by the existence of a rule $\boldsymbol{A}^- \rightarrow \boldsymbol{C}$, where $\boldsymbol{A}^- \subset \boldsymbol{A}$. This is because $\boldsymbol{A} \rightarrow \boldsymbol{C}$ may have higher confidence than $\boldsymbol{A}^- \rightarrow \boldsymbol{C}$. Consider the example in Table 3. In this case, having two items in the antecedent $\{C1, C2\}$ forms a stronger implication of $C3$ than either $C1$ or $C2$ acting alone.

As a reminder from Equation (5), $\sigma(\{C1, C2\})$ counts all observations where $C1$ and $C2$ are simultaneously violated. This is effectively the intersection of the infeasible region defined by the constraints (where both items occur). However, if the goal is to remove constraints, it is not important if $\{C1, C2\}$ forms an implication of $C3$. Neither $C1$ nor $C2$ can be removed, as $C1$ and $C2$ restrict other areas as well. In order to remove either $C1$ or $C2$ (due to $C3$), rules $C1 \rightarrow C3$ or $C2 \rightarrow C3$ would be required. In this case, the confidence of these rules is only 67%. Thus, neither is redundant due to $C3$ (if minconf > 0.67). This example shows that antecedents with more than one item do not form patterns/rules that can be acted on, and do not need to be considered.

Table 3. Example of increasing confidence with additional items in the antecedent.

$C1$	$C2$	$C3$	$Conf(\{C1, C2\} \rightarrow C3)$	$Conf(C1 \rightarrow C3)$	$Conf(C2 \rightarrow C3)$	
1	1	1				
1	1	1	$\frac{\sigma(C1C2C3)}{\sigma(C1C2)} = \frac{2}{2} = 100\%$	$\frac{\sigma(C1C3)}{\sigma(C1)} = \frac{2}{3} = 67\%$	$\frac{\sigma(C2C3)}{\sigma(C2)} = \frac{2}{3} = 67\%$	
0	0	1				
1	0	0				
0	1	0				

3.3.2. *Restricting association rules to one-item consequents*

Another simple way to limit the number of rules is to limit consequents to contain exactly one item. It can be shown that if a rule exists with a k-item consequent ($k > 1$), then one-item consequent rules also exist for those items. This is inherent to the fact that *a priori* generates rules from consequents with fewer items and builds on those to form new rules. For completeness, a simple theorem and proof is provided below.

THEOREM 3.2 *Given a consequent* $\boldsymbol{C}$*, antecedent* $\boldsymbol{A}$ *and subset* $\boldsymbol{C}^- \subseteq \boldsymbol{C}$*, if* $c(\boldsymbol{A} \to \boldsymbol{C}) \geq$ minconf, *then* $c(\boldsymbol{A} \to \boldsymbol{C}^-) \geq$ minconf.

Proof

$$c(\boldsymbol{A} \to \boldsymbol{C}) = \frac{\sigma(\boldsymbol{AC})}{\sigma(\boldsymbol{A})} \leq \frac{\sigma(\boldsymbol{AC}^-)}{\sigma(\boldsymbol{A})} \text{ multiply both sides by } \sigma(\boldsymbol{A}),$$

$$\sigma(\boldsymbol{AC}) \leq \sigma(\boldsymbol{AC}^-) \text{ by the anti} - \text{monotonic property of } \sigma$$

$$\therefore \text{ if } c(\boldsymbol{A} \to \boldsymbol{C}) = \frac{\sigma(\boldsymbol{AC})}{\sigma(\boldsymbol{A})} \geq \text{minconf}, \; c(\boldsymbol{A} \to \boldsymbol{C}^-) = \frac{\sigma(\boldsymbol{AC}^-)}{\sigma(\boldsymbol{A})} \geq \text{minconf}$$

■

To limit the mining process to one-item consequents *and* one-item antecedents, the confidence-pruning stage is only executed for the frequent itemsets with two items, which greatly improves efficiency. Note that the larger frequent itemsets are still required for co-occurrence identification.

3.4. *Mining-covered constraints and additional information (Step 5)*

The next step is to find redundancy rules for constraints that do not co-occur and are not implicit. This can be accomplished by looking for constraints which are never violated alone. According to Boneh's (1984) main theorem, if an observation violates only one constraint, that constraint is absolutely necessary. Conversely, it is clear that if a constraint is never violated alone, it is redundant as it has no unique effect on the feasible space. Consider the case in Table 4. The union of violations in $C1$ and $C3$ covers the observations violated in $C2$. If $C2$ were removed from set of constraints, the feasible points would remain unchanged for the given data. This makes $C2$ redundant (by covering). Determining whether a constraint is covered is fundamentally different from identifying co-occurrence or implication. Namely, to check covering, additional

Table 4. Covered constraint example.

$C1$	$C2$	$C3$
1	0	0
1	0	0
1	1	0
1	1	0
0	1	1
0	1	1
0	0	1
0	0	1

Table 5. Covered constraint example with duplicate observations removed.

$C1$	$C2$	$C3$
0	0	1
0	1	1
1	0	0
1	1	0

scans of the data are needed. However, note that observations do not need to be counted as when checked for co-occurrence or implication. Therefore, checking whether a constraint is covered only requires the set of unique observations (as in the SC approach; Boneh 1984). This reduced set of observations is shown in Table 5, which is generated from Table 4 by eliminating the repetitions.

As explained by Theorem 12 of Boneh's (1984) paper: if there is a case where constraint $C_k \in \boldsymbol{C_T}$ is violated, and all other constraints $\boldsymbol{C}\backslash\{C_k\}$ are satisfied, then C_k is necessary. In this article, $\boldsymbol{C}_T$ is the set of non-co-occurring, non-implicit, frequent constraints and $\boldsymbol{C}$ is the set of all constraints. In other words, covering is only checked for constraints which are not already explained by co-occurrence or implication. A simple scan to find a case where C_k is violated alone is executed for each $C_k \in \boldsymbol{C_T}$ (on the set of unique observations). If no unique violation case exists, the constraint does not uniquely impact the overall feasibility of the problem, and a rule is saved. In Table 5, there is no case where $C2$ violation occurs alone. Thus, $C2$ is covered.

3.5. *Additional rule types for constraint mining*

In addition to finding bundles and association rules, two more special cases are added for constraint analysis.

3.5.1. *Infrequent constraints*

It is often useful for the designer to understand and review constraints which are mostly passive, especially if they are costly to evaluate during optimization. Infrequent constraints are those which are rarely (or never) violated in the sample of observations (as quantified by support). Sequentially, infrequent constraint rules are identified and saved as $s(\boldsymbol{I})$ is computed. This way, determining infrequent constraints does not require additional scans of the constraint data.

DEFINITION 3.2 (INFREQUENT CONSTRAINT): *An individual constraint C_i is infrequent if* $s(C_i) \leq \text{minsupp}$.

3.5.2. *Infeasible problems*

An infeasible problem does not contain any feasible area. If a problem is infeasible, it may require reformulation (*e.g.* making a restrictive constraint an objective) or an optimization algorithm that specializes in highly constrained problems. To search for overall feasibility, the *any* operator (γ) is used on each observation $\boldsymbol{e}_i$ in the unique set of observations. If any $\gamma(\boldsymbol{e}_i)$ is 0, a feasible point exists, the search is stopped, and the problem is considered feasible. If no feasible point exists, the problem is deemed infeasible.

DEFINITION 3.3 (INFEASIBLE PROBLEM): *A problem is infeasible if there is no observation e_i in the unique set of observations such that $\gamma(\boldsymbol{e}_i) = 0$.*

4. Collecting data and setting algorithm parameters

4.1. *Collecting constraint data*

To collect data for constraint mining, constraint checks are randomly generated in an iterative fashion. The number of constraint checks to generate each iteration is given by Equation (6):

$$nS_i = m * nv \tag{6}$$

where nS_i is the number of samples generated per iteration; m is a constant chosen by the user; and nv is the number of design variables in the problem. The parameter m should be chosen based on the computational intensity and number of constraints. For problems with computationally inexpensive constraints, thousands of samples can be tested (*e.g.* $m = 1000$). For computationally expensive constraints, fewer may be budgeted. Each iteration, nS_i random samples are evaluated on the constraints, and the result is added to the table of Boolean observations. Rules are then mined using the method described in this article. The stopping criteria for the iterative process are a number of iterations (*e.g.* 5) have occurred without a change in rules, or that a maximum number of total observations have been sampled. Figure 5 shows the data-generation process, where the number of iterations with the same rules is denoted by nI, and the limit on iterations with no rule change is IL. The total number of samples is given by nT, and the total sample limit is given by TL. Both IL and TL are chosen prior to running the method.

It is important to note that there is no theoretical guarantee on the number of observations required to establish correct rules. In fact, even if the conditional probability of a rule is 100%, based on the limited sample data, it may be found to be incorrect owing to missing unique observations. This is explained by examples in Section 5, and is an inherent limitation of the probabilistic method presented. Boneh (1984) provides a brief study on determining an appropriate number of constraint checks. He makes two assumptions: (1) the probability of violating a constraint is known *a priori*; and (2) the probability of violation is equal for all constraints (Boneh 1984). In practice, however these assumptions do not hold. Overall, if constraints are suppressed based on the rules found, it is suggested to confirm that the final solution is feasible by testing it against all constraints (including those which were suppressed).

The efficiency costs or savings of preoptimization redundancy identification are heavily dependent on the design problem, and on the algorithm used for optimization. For instance, some stochastic algorithms such as mode pursuing sampling (MPS) (Wang, Shan, and Wang 2004) operate under the assumption that constraints are cheap to evaluate, and generate many cheap points for each iteration. Subsequently, a subset of feasible designs is chosen to be evaluated with the objective function. If the cheap constraint assumption is not valid for a particular problem, it may be beneficial to check for redundancies prior to a lengthy optimization process. Of

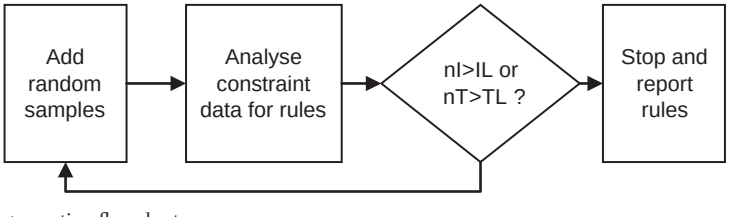


Figure 5. Data generation flowchart.

course, evaluating constraints before optimization also requires sampling, as explained above. A study has not been conducted to evaluate the efficiency trade-off of preoptimization redundancy checking for a set of constrained computationally intensive black-box problems, as both the problems and algorithms must be considered. Therefore, this method should be considered as a method to gain information about the problem as formulated, as opposed to a method to improve optimization efficiency.

4.2. *Setting the rule parameters*

Identifying which constraints co-occur and which are implicit is useful to gather information about the relationships between constraints. In engineering practice, however, a single observation that contradicts a redundancy is sufficient to justify the need for both constraints. If a constraint is suppressed that was not completely redundant, the solution found from optimization may be infeasible. Therefore, minjacc and minconf may be lowered to see which constraints overlap for insight, but they should be set to 1 if used for suppressing constraints.

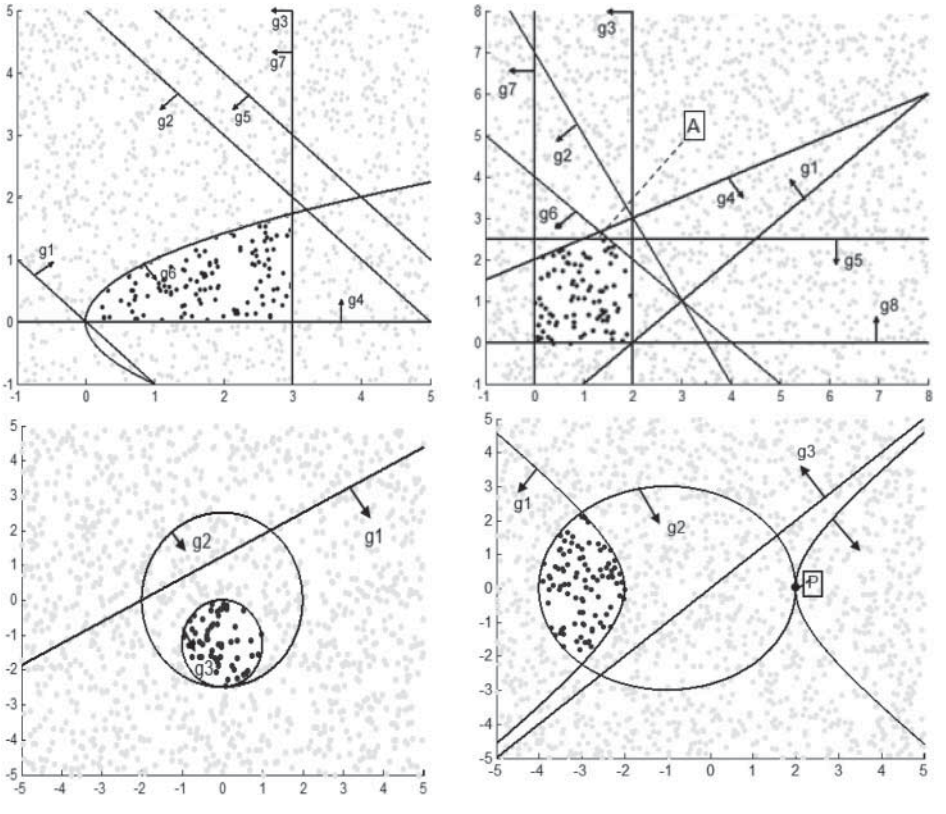


Figure 6. Example 1 (top left) with 1200 samples; Example 2 (top right) with 1600 samples; Example 3 (bottom left) with 1200 samples and Example 4 (bottom right) with 1200 samples. Dark points are feasible. The axes for all plots are $x2$ vs $x1$.

Table 6. Example 1.

$g1: -x1 - x2 \leq 0$
$g2: x1 + x2 \leq 5$
$g3: x1 \leq 3$
$g4: -x2 \leq 0$
$g5: x1 + x2 \leq 6$
$g6: -x1 + x2^2 \leq 0$
$g7: 2(x1) \leq 6$

Table 7. Example 2.

$g1: x1 - x2 \leq 2$
$g2: 2(x1) + x2 \leq 7$
$g3: x1 \leq 2$
$g4: -x1 + 2(x2) \leq 4$
$g5: 2(x2) \leq 5$
$g6: x1 + x2 \leq 4$
$g7: -x1 \leq 0$
$g8: -x2 \leq 0$

Table 8. Example 3.

$g1: -x1 + 2(x2) \leq 4$
$g2: x1^2 + (x2 - 1)^2 \leq 4$
$g3: x1^2 + x2^2 \leq 1$

Table 9. Example 4.

$g1: -x1^2 + x2^2 \leq -4$
$g2: (x1 + 1)^2 + x2^2 \leq 9$
$g3: x1 - x2 \leq 0$

5. Examples and results

5.1. *Two-variable mathematical examples*

5.1.1. *Problem definition*

In this section, constraint mining is tested on two-dimensional mathematical examples which can be easily visualized as in Figure 6. The arrows in Figure 6 indicate the direction of feasibility. The first problem has seven constraints, including one nonlinear constraint. The second problem is from Telgen (Karwan *et al*. 1983, 57), including eight linear constraints. The third and fourth problems are from Feng's (1999) analysis of the set-covering method. The problems are defined algebraically in Tables 6–9, respectively. In Example 1, it is clear that constraints $g3$ and $g7$ are multiples of one another, meaning that they represent the same constraints. Furthermore, $g2$ is a tighter restriction than $g5$. In Example 2, the patterns are less obvious, but can be seen in Figure 6. Namely, $g2$ has no influence on the feasible space, owing to the combination of $g6$ and $g3$. In Example 3, the constraints $g1$ and $g2$ are clearly redundant. In Example 4, $g3$ appears redundant, but it will be explained that it may be necessary.

Table 10. Example 1 rules.

Rule	Rule conditional probability	Rule type
'$g3$ and $g7$ co-occur'	100%	Co-occurrence
'$g5$ is redundant to $g2$ by implication'	100%	Implication
'$g1$ is covered due to the union of other constraints'	100%	Covering

Table 11. Example 2 rules.

Rule	Rule conditional probability	Rule type
'$g1$ is covered due to the union of other constraints'	100%	Covering
'$g2$ is covered due to the union of other constraints'	100%	Covering
'$g5$ is covered due to the union of other constraints'	100%	Covering

Table 12. Example 3 rules.

Rule	Rule conditional probability	Rule type
'$g1$ is redundant due to $g3$ by implication'	100%	Association
'$g2$ is redundant due to $g3$ by implication'	100%	Association

Table 13. Example 4 rules.

Rule	Rule conditional probability	Rule type
'$g3$ is covered due to the union of other constraints'	100%	Covering

5.1.2. *Constraint mining results*

The results from constraint mining are presented in Tables 10–13. The settings assume that a single contradictory case is sufficient to discredit a rule; therefore, minsupp $= 0$ and minjacc $=$ minconf $= 1$. In Example 1, 1200 samples were tested. In Example 2, 1600 samples were tested. The number of iterations without a rule change was set to 5, and m was set to 100 (thus $nSi = 200$). The results from Tables 10 and 12 are correct by inspection of 6. However, the results from Tables 11 and 13 illustrate the probabilistic nature of the method. In Table 11, the rule stating that $g5$ is covered is false. Indeed, there is an area (labelled **A** in 6) that is uniquely restricted by $g5$. Unfortunately, no sample was generated in that region during random sampling. Consequently, from the constraint data, $g5$ has no case where it is violated and others are not; it is covered based on the collected observations. Similarly, in Table 13, the rule that $g3$ is redundant is possibly incorrect. There is a single point $\boldsymbol{P}$, at the junction of $g2$ and $g1$, which is feasible. Feng (1999) refers to this as a quasi-minimizer surface point. The constraint $g3$ restricts $\boldsymbol{P}$, but it is extremely unlikely to sample $\boldsymbol{P}$ (or any particular single point) by random sampling. This illustrates the pitfall of using a probabilistic method: it is possible to miss rare but possibly important restrictions placed by constraints which are seemingly redundant. On the other hand, this also supports the idea of presenting readable information that a designer can validate (the focus of this article), as opposed to *automatically* removing constraints prior to optimization. Furthermore, for black-box problems there is no deterministic method to mine constraint redundancy which guarantees correctness.

Table 14. Rules for continuous pressure vessel design optimization.

Rule	Rule conditional probability	Rule type
'$g2$ is redundant due to $g1$ by implication'	99.6%	Implication

Table 15. Sensitivity of parameter m for the pressure vessel example.

m	No. of runs with correct rules	No. of constraint checks on average per run
10	10/10	256
20	10/10	544
50	10/10	1200
100	10/10	2400

5.2. *Pressure vessel design example*

In this section, the design of a pressure vessel is used as a test problem. This problem is an example in globally optimal design by Wilde (1978), and has been used as a benchmark problem in the engineering optimization literature (Lewis and Mistree 1996; Wang, Shan, and Wang 2004). The problem has four variables: tank radius (R), tank length (L), shell thickness (T_s) and head thickness (T_h). The objective is to minimize the aggregated system cost, including materials and manufacturing. There are three constraints in the original problem, based on American Society of Mechanical Engineers (ASME) pressure vessel standards, and one objective cost function. Although the problem is often posed with discrete thickness values, in this article, the problem is defined for continuous values, as used by Wang, Shan, and Wang (2004), with Equations (7)–(10):

$$\min f(X) = 0.06224\, T_s RL + 1.7781\, T_h R^2 + 3.1661\, T_s L + 19.84\, T_s^2 R \tag{7}$$

subject to:

$$g1 = 0.0193R - T_s \leq 0 \tag{8}$$

$$g2 = 0.00954R - T_h \leq 0 \tag{9}$$

$$g3 = 1296000 - \pi R^2 L - \left(\frac{4}{3}\right)\pi R^3 \leq 0 \tag{10}$$

The bounds for the variables are as follows (each variable is in inches):

$$R \in [25, 150],\ T_s \in [1, 1.375], L \in [25, 240], T_h \in [0.625, 1]$$

For this problem, 1200 samples were generated, with minsup = 0.1, minjacc = minconf = 0.9. For data generation, the parameter m is set to 50 ($nSi = 200$). The resulting list of rules is shown in Table 14. The algorithm was also repeated 10 times, with different values for m to test the sensitivity of the method with respect to the number of samples. It was found that even with $m = 10$ ($nSi = 40$), the rule is the same each time the method is tested for this problem. This results in an average number of 256 samples per run, as opposed to using 1200 (Table 15).

Surprisingly, it was found that violations in $g2$ imply $g1$ in approximately 99% of samples. Therefore, constraint $g2$ could be suppressed (with the risk that the optimum falls into the 0.4% of cases where the rule is known to be false). Optimization was subsequently performed with and without $g2$ using the MPS algorithm (Wang, Shan, and Wang 2004). The result in both cases is

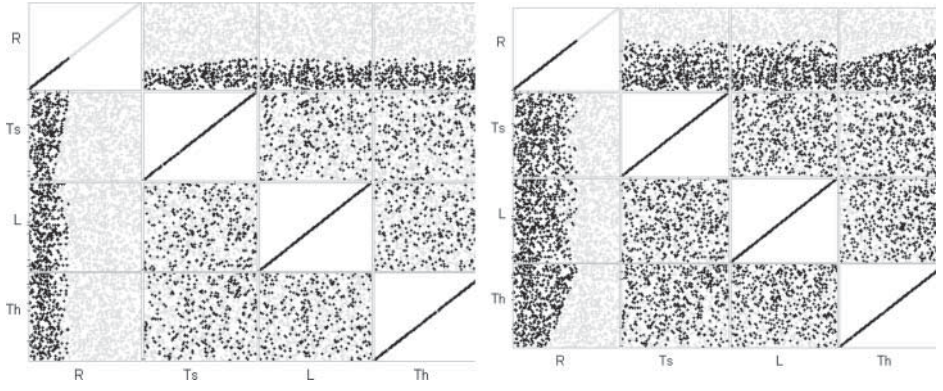


Figure 7. Constraints $g1$ (left) and $g2$ (right) as scatterplot matrices. Dark points are feasible.

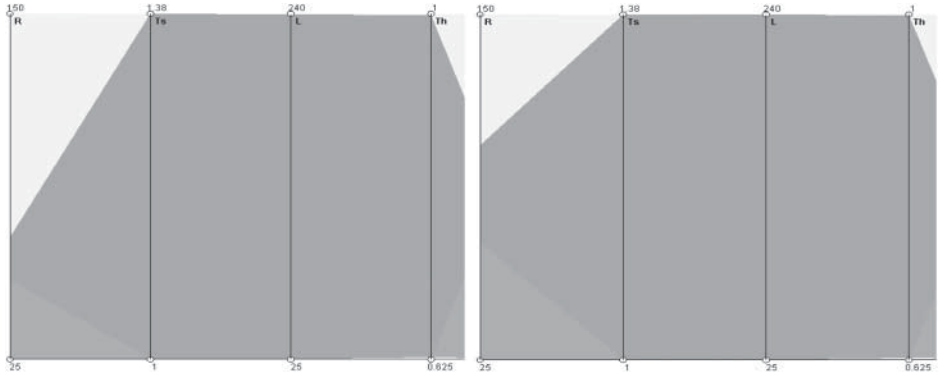


Figure 8. Constraints $g1$ (left) and $g2$ (right) as polygons in parallel coordinates. The dark area is feasible.

$f^* = 7006.8$, at $R^* = 51.8''$, $T_s^* = 1''$, $L^* = 84.6''$ and $T_h^* = 0.625''$. This suggests that $g2$ really has no effect on the optimum.

In general, there is no guarantee that the solution found with $g2$ suppressed will be the same as with $g2$ unsuppressed. As stated previously, the rules are based only on the constraint data sampled; even if the conditional probability is 100%, there may be a contradictory case that has not been sampled. Therefore, it is recommended to treat these rules as informative guidelines about the problem formulation, and to verify the final answer with all constraints that make physical sense.

To verify association rules beyond two dimensions, the underlying data can be shown with multivariate visualization. Two methods of visualization are presented, in Figures 7 and 8. In both the scatterplot matrices and parallel coordinates plots below, it is clear that the restriction of $g1$ covers the same region as $g2$ but is stricter on the variable R.

6. Conclusion

This article presents a method to find redundant constraint groups in black-box optimization problems. Association analysis, from data mining, is applied for constraint redundancy

identification, which is a new application for the method. Rather than directly applying association analysis, this work developed a new method and theorems for constraint redundancy identification, which incorporates association analysis techniques. In summary:

(1) A systematic procedure of applying association analysis to constraint redundancy identification is developed.
(2) This article proposed a new method to find constraints that co-occur, using Jaccard similarity, before performing association analysis on the remaining frequent itemsets. This prevents the generation of many unnecessary association rules.
(3) Additional limitations were added to *a priori* rule generation for the application. In particular, it was shown that implication rules with more than two items do not provide additional information for constraint redundancy identification. Therefore, the association rule generation phase can be restricted to two itemsets which do not co-occur.
(4) Additional redundancies (due to covering) are checked on the remaining itemsets using a reduced set of observations as in the set-covering approach.
(5) The result of the proposed method is a set of readable rules that the user may choose to act on, as opposed to a reduced set of constraints as in other redundancy identification methods.

The method was first applied to mathematical problems. It was found that the rules summarize the relationship among constraints. However, it was also pointed out that, because of the method's probabilistic nature, the accuracy of the rules depends on the sample points. An incorrect rule was mined owing to a lack of sampling in a particular region. The method was then applied to an engineering pressure vessel design problem. It was found that a design constraint for this benchmark problem is likely to be redundant. The rules were validated using multivariate visualization. Although the presented method cannot guarantee correctness, constraint rules provide design engineers with new information on how their constraints restrict the design space. Ultimately, this may lead to improved problem formulations.

References

Agrawal, Rakesh, Tomasz Imielinksi, and Arun Swami. 1993a. "Database Mining: A Performance Perspective." *IEEE Transactions on Knowledge and Data Engineering* 5 (6): 914–925. doi:10.1109/69.250074

Agrawal, Rakesh, Tomasz Imielinski, and Arun Swami. 1993b. "Mining Association Rules between Sets of Items in Large Databases." In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, 207–216. New York, NY: ACM Press.

Atluri, Gowtham, Rohit Gupta, Gang Fang, Gaurav Pandey, Michael Steinbach, and Vipin Kumar. 2009. "Association Analysis Techniques for Bioinformatics Problems." *Lecture Notes in Computer Science* 5462 (0302): 1–13. doi:10.1007/978-3-642-00727-9_1

Berbee, H. C. P., C. G. E. Boender, A. H. G. Rinnooy Ran, C. L. Scheffer, R. L. Smith, and Jan Telgen. 1987. "Hit-and-Run Algorithms for the Identification of Nonredundant Linear Inequalities." *Mathematical Programming* 37 (1): 184–207. doi:10.1007/BF02591694

Boneh, Arnon. 1984. "Identification of Redundancy by a Set-Covering Equivalence." In *Operational Research" 84. Proceedings of the Tenth International Conference on Operational Research*, edited by J. P. Brans, 407–422. Amsterdam: North Holland/Elsevier.

Boot, T. C. G. 1962. "On Trivial and Binding Constraints in Programming Problems." *Management Science* 8 (4): 419–441.

Brearley, A. L., G. Mitra, and H. P. Williams. 1975. "Analysis of Mathematical Programming Problems Prior to Applying the Simplex Algorithm." *Mathematical Programming* 8 (1): 54–83. http://www.jstor.org/stable/2627245

Caron, R. J., J. F. McDonald, and C. M. Ponic. 1989. "A Degenerate Extreme Point Strategy for the Classification of Linear Constraints as Redundant or Necessary." *Journal of Optimization Theory and Applications* 62 (2): 225–237. doi:0.1007/BF00941055

Cheng, Hong, Philip Yu, and Jiawei Han. 2006. "AC-Close: Efficiently Mining Approximate Closed Itemsets by Core Pattern Recovery." In *Proceedings of the Sixth International Conference on Data Mining (ICDM'06)*, 839–844. Hong Kong: IEEE.

Feng, Jinghua. 1999. "Redundancy in Nonlinear Systems: A Set Covering Approach." Masters Thesis. University of Windsor.

Han, Jiawei, Jian Pei, Yiwen Yin, and Runying Mao. 2004. "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach." *Data Mining and Knowledge Discovery* 8 (1): 53–87. doi:10.1023/B:DAMI.0000005258.31418.83

Huang, Wenxue, Milorad Krneta, Limin Lin, Jianhong Wu, and Generation5 Math Technologies. 2006. "Association Bundle—A New Pattern for Association Analysis." In *Sixth IEEE International Conference on Data Mining-Workshops (ICDMW'06)*, 601–605. Hong Kong: IEEE.

Karwan, Mark H., Vahid Lofti, Jan Telgen, and Stanley Zoints. 1983. *Redundancy in Mathematical Programming: A State-of-the-Art Survey*. New York: Springer.

Lewis, Kemper, and Farrokh Mistree. 1996. "Foraging-Directed Adaptive Linear Programming: An Algorithm for Solving Nonlinear Mixed Discrete/continuous Design Problems." In *Proceedings of the 1996 ASME Design Engineering Technical Conferences and Computers in Engineering Conference*, 1601. Irvine, CA.

Özden, Banu, Sridhar Ramaswamy, and Avi Sillberschatz. 1998. "Cyclic Association Rules." In *14th International Conference on Data Engineering, 1998 Proceedings*, 412–421. Orlando, FL: IEEE.

Pasquier, Nicolas, Yves Bastide, Rafik Taouil, and Lofti Lakhal. 1999. "Discovering Frequent Closed Itemsets for Association Rules." In *ICDT "99 Proceedings of the 7th International Conference on Database Theory*, edited by Catriel Beeri and Peter Buneman, 398–416. Jerusalem, Israel: Springer.

Paulraj, S., and P. Sumathi. 2010. "A Comparative Study of Redundant Constraints Identification Methods in Linear Programming Problems." *Mathematical Problems in Engineering* 2010 (723402): 1–16. doi:10.1155/2010/723402

Potter, C., S. Klooster, M. Steinbach, P. Tan, V. Kumar, S. Shekhar, R. Nemani, and R. Myneni. 2003. "Global Teleconnections of Ocean Climate to Terrestrial Carbon Flux." *Journal of Geophysical Research: Atmospheres* 108 (D17): ACL 12-1–ACL 12-12. doi:10.1029/2002JD002979

Tan, Pang-Ning, Vipin Kumar, and Jaideep Srivastava. 2002. "Selecting the Right Interestingness Measure for Association Patterns." In *KDD" 02 Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 232–241. Edmonton, AB: ACM Press.

Tan, Pang-Ning, Michael Steinbach, and Vipin Kumar. 2006. *Introduction to Data Mining*. Boston: Pearson Education.

Thompson, Gerald L., Fred M. Tonge, and Stanley Zionts. 1966. "Techniques for Removing Nonbinding Constraints and Extraneous Variables from Linear Programming Problems." *Management Science* 12 (7): 588–608. doi:10.1287/mnsc.12.7.588

Wang, Liqun, Songqing Shan, and G. Gary Wang. 2004. "Mode-Pursuing Sampling Method for Global Optimization on Expensive Black-Box Functions." *Engineering Optimization* 36 (4): 419–438. doi:10.1080/03052150410001686486

Wilde, Douglass. 1978. *Globally Optimal Design*. New York, NY: John Wiley and Sons.

Xiong, Hui, Pang-Ning Tan, and Vipin Kumar. 2006. "Hyperclique Pattern Discovery." *Data Mining and Knowledge Discovery* 13 (2): 219–242. doi:10.1007/s10618-006-0043-9

Zhou, Ling, and Stephen Yau. 2007. "Efficient Association Rule Mining among Both Frequent and Infrequent Items." *Computers & Mathematics with Applications* 54 (6): 737–749. doi:10.1016/j.camwa.2007.02.010