Rough Sets and Concept Lattices

G. Deon Oosthuizen*
Department of Computer Science, Stanford University
Stanford, California CA94305, USA

Abstract
A Concept lattice is a special kind of lattice, the explicit representation of which can be viewed as a semantic network with special properties. Concept lattices have been applied to Machine Learning as well as to the uncovering of the underlying structure in discrete-valued data. They also embody the cladistic approach to classification. This paper describes how the use of a concept lattice as representation model is related to the rough set approach to data analysis and how operations of rough set theory can be implemented using a concept lattice.

1 Introduction
In the course of the human endeavour to gain insight from large data bases, the uncovering of dependence and causality has been studied by many people. The use of graphs to model dependence and causality in data has been shown to have several advantages [12]. It provides a vivid and concise account of the relations between the variables in the universe of discourse.

In this paper we consider the use of a special kind of data structure for the distillation of knowledge from data: a formal lattice. Although Wille [15] introduced the notion of a concept lattice, the particular organization of data described here also corresponds to the so-called cladistic approach to classification used by biologists and linguists for some time [5]. In fact, the underlying principle referred to as the duality of intension and extension had already been noted by Aristotle [14] and refers to the inverse relation between the number of properties required to define a concept and the number of entities to which the concept applies. This paper briefly explains how a concept lattice can be applied to discover regularities in data, i.e. to learn rules. It then informally describes how concept lattices relate to rough set theory, explaining that a concept lattice constitutes a representation model for the implementation of rough set operators.

We first describe how a concept lattice is constructed, how dependence relationships and rules are derived and, finally, its relation to rough set theory.

2 Concept Lattices
Since we are concerned with the graphical representation of a formal lattice (referred to as a Hasse diagram) we provide a graph theoretical description for it. A lattice is a directed acyclic graph with an additional constraint: every pair

*On leave from the University of Pretoria, Pretoria, 0001, South Africa.
of nodes in the graph has a unique nearest common descendent - or meet - and a unique nearest common ancestor - their join. (In the formal mathematical definition these are referred to as greatest lower bounds and least upper bounds respectively.)

The lattices discussed here are of a special kind, called concept lattices [15] [7], which have the following additional properties:

- Apart from the nodes adjacent to the universal node (at the top) and the NULL node at the bottom, no nodes in the graph have exactly one parent or exactly one child.

- No node has a parent (i.e. no node is directly linked to another node) to which it is also indirectly linked by means of a path that goes via one or more other nodes.

Concept lattices have also been referred to as Galois lattices in the literature [2].

In the current application of concept lattices, we treat them as semantic networks, reflecting entity-attribute relationships. For this reason the universal node (at the top) which ensures that all pairs of nodes in a lattice have joins and the NULL node (at the bottom) which ensures that all pairs of nodes have meets, are omitted. Consequently, every pair of nodes in the current lattices has a unique meet or no meet, as well as a unique join or no join. This is merely an implementation measure, however, and has no impact on the fundamental properties of the lattices. Thus, the concept lattices described here consist of a set \( A \) of nodes with no parents, called attributes, a set \( E \) of nodes with no children, called entities, and a set \( I \) of nodes in between, called internal nodes.

A lattice is constructed using discrete-valued entities as input, i.e. entities consisting of vectors of discrete attribute-values (like a record in a database). The graph is constructed by creating a node (an element of \( E \)) for each entity, and connecting it to each of its attributes by means of upward pointing arcs, whilst ensuring that the graph remains a lattice, i.e. ensuring that each pair of attributes or internal nodes has a unique meet - if this condition is met, all pairs of nodes automatically have unique joins. During this process, the elements of \( I \) are created between the bottom and top rows \(^1\). It can be shown that a given set of entities gives rise to a unique lattice.

Each node in \( I \) relates a unique subset of \( E \) to a unique subset of \( A \). Intuitively, the elements of \( I \) can be regarded as sets of entities that have attributes in common. With each node, we associate a value referred to as its strength, reflecting the number of entities below it in the graph. The strength of a node could also be interpreted as the size of the set (or class) denoted by the node, i.e. the number of entities which have the attributes above the node in common. We shall henceforth refer to the attributes above a given node as attributes spanned by the node, and to the entities below it as entities covered by it.

In the worst case, the number of nodes in the lattice increases exponentially with the number of attributes per entity. However, as we will explain later, the lattices can be pruned to prevent this. The computational complexity of constructing a pruned lattice can be kept at a level less than \( O(n^3) \), where

\(^1\)The manner in which this is accomplished is not relevant to the topic of this paper. Algorithms are described in [8][1].