OPUS: An Efficient Admissible Algorithm for Unordered Search

Paper for Subgroup Discovery Seminar 2017

Henrik Jilke

Abstract

Algorithms that provide a fast search through vast spaces are of great interest. In particular, in tasks such as subgroup discovery it is important to traverse the search space efficiently. Usually, this is achieved via excluding parts of the search space. In this work we focus on *OPUS*, a set of branch and bound search algorithms that optimize the effect of any pruning in unordered search spaces. Given an admissible function, *OPUS* guarantees to find an optimal solution. Recently employed methods fail either to find optimal states, do not prune optimally, or have unreasonable time and memory consumptions. In contrast, *OPUS* allows for optimal pruning with negligible computational overhead. Furthermore, the presented work introduces additional heuristics to further enhance the effect of pruning. I review *OPUS* as applied to subgroup discovery and classification tasks. The results show that this algorithm can even efficiently optimize over search spaces of size 2^{162}.

1 Introduction

A great deal of applications in the area of computer science rely on solving a searching problem. Especially, certain problems in machine learning and artificial intelligence, such as subgroup discovery (SGD), rule based learning, and frequent item set mining, can be formulated as a searching problem. In particular, subgroup discovery [see Atzmueller, 2015] searches for subgroups of a given population that are statistically unusual with respect to the attribute of interest [Schmidt et al., 2010].
As in many other problems, the search spaces are huge, typically exponential in the input. Therefore, algorithms which consider each element of the search space, are not suitable for practical applications, neither in runtime nor in memory requirements. One way to tackle this problem is to disregard areas of the search space in which we can not find a solution, using pruning techniques. Other approaches try to prune parts of the search space based on heuristics, which can be faster, but might include the pruning of goal states. Beam search, for example, uses heuristics to expand a fixed number of promising nodes [Norvig, 1992]. However, beam search does not guarantee to find an optimal solution, and might exclude possible goal states. An other class of algorithms

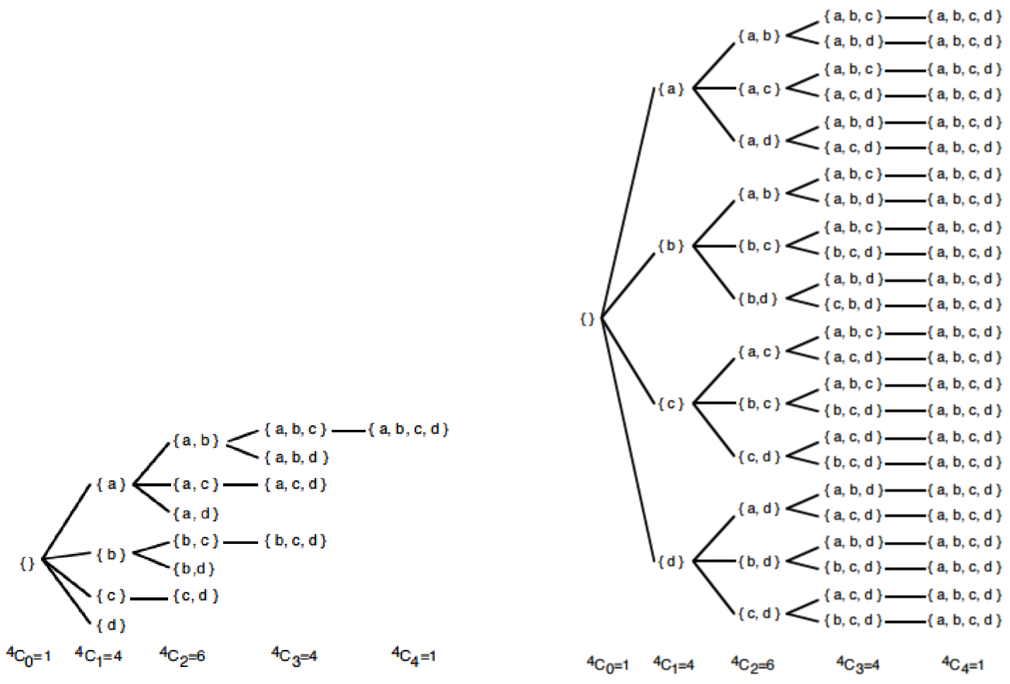
performs a fixed ordered, admissible search, that is, they guarantee an optimal solution if there is one [Clearwater and Provost, 1990, Schlimmer et al., 1993, Segal and Etzioni, 1994]. Nevertheless, the fixed ordering of the search space may have drawbacks in the amount of pruned space. Additionally, pruning algorithms based on fixed ordered search, such as the approach in [Schlimmer et al., 1993], suffer major computational memory issues, resulting from the need to store all pruned states. Contrary, the Feature Subset Selection algorithm (FSS) [Narendra and Fukunaga, 1977], a branch and bound search algorithm, organizes the search space dynamically, using heuristics, in order to achieve a better pruning. Generally, the FSS algorithm is bounded to a single optimal solution and requires that the value of a node does not increase as a result of an application of an operator.
In order to overcome these problems, Webb [1995] generalizes the FSS algorithm to multiple optimal solutions, the values of the states are less restricted and additional pruning rules are included. He develops $OPUS$, a set of search algorithms, that optimize the effect of any pruning in unordered search spaces. The results show that his $OPUS$ algorithm allows even the search in spaces of size 2^{168}.

2 Preliminaries

Consider an itemset mining task with a ground set of items $X = \{a, b, c, d\}$. The smallest search space considering all items is a tree of size $2^{|X|}$, see figure 1 (a). A naive search space, that does not consider duplicates, would result in a tree of size $\sum_{i=1}^{|X|} \frac{|X|!}{(|X|-i)!}$, see figure 1 (b).
Generally, we can group the search spaces into two types: unordered search space and ordered search space. In an ordered search space, the order of the application of the operators matters. Consider a language generation system, the order of words (operators) matters as long as there are grammar rules in the background. Apart from that, there are a lot of applications that search through an unordered search space, in which the order of the concatenation of the operator does not matter. If we describe subgroups as a conjunction of basic propositions (CNF), it is irrelevant in which order we arrange the basic propositions.
Search algorithms like $OPUS$ or A^* aim to traverse the search space in a suitable way, in order to find goal states. Both algorithms guarantee to find a goal state if the input heuristic given to the algorithm is admissible. A heuristic is admissible if it does not overestimate the cost to reach the goal state [Pearl, 1984].

3 Contribution

The $OPUS$ algorithms were designed to enable an efficient search for nodes, that optimize some function Webb [2001]. The algorithms guarantee to maximize the effect of any pruning of the search space, using optimistic pruning and other pruning techniques [Webb, 1993]. They consider every state at most ones and seek to identify search operators that can be disregarded below the search tree of a node.



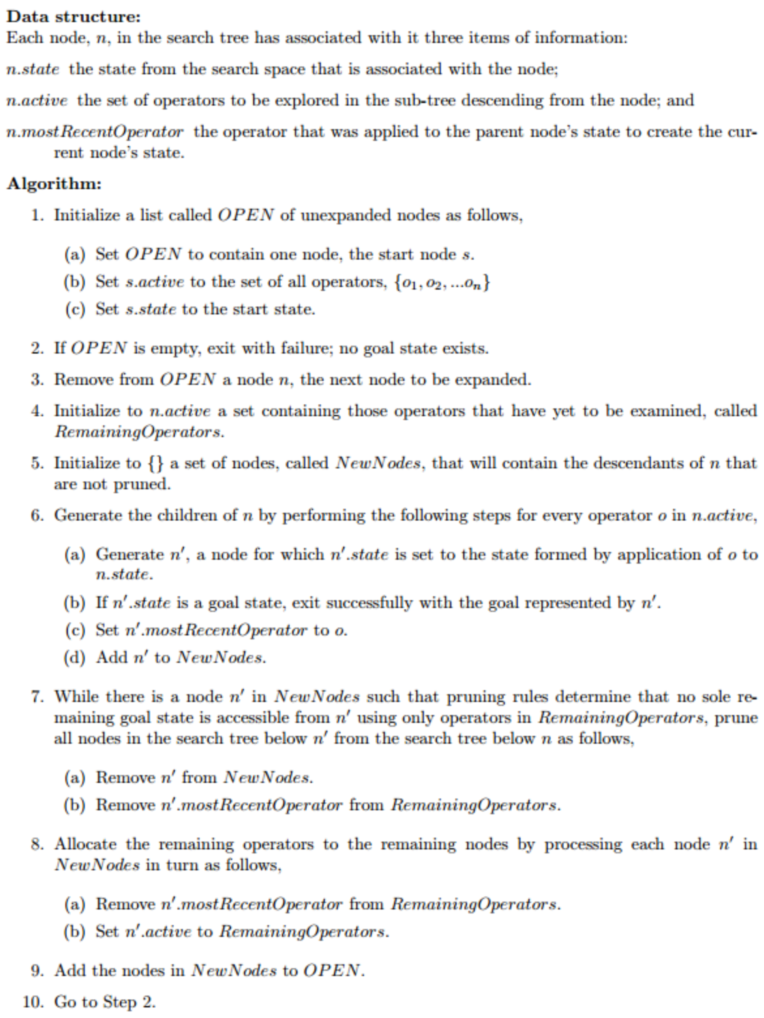
(a) fixed-ordered search space

(b) search space with duplicates

Figure 1. A comparison of different search spaces, and the effect of the generation of duplicates.

Webb [1995] outlines two algorithms: the $OPUS^s$ and the $OPUS^o$ algorithm. The pseudocode of the $OPUS^o$ algorithm can be seen in figure (2). In $OPUS^s$ every node n has a list $n.active$, which contains the list of remaining operators for this node. Additionally, each node has a state $n.state$ that describes the operators that had been applied up to the node n (e.g $n.state = \{abc\}$). The decision to store this information disallows, on one hand, generating duplicates, and on the other allows for pruning operators in the subtree below a node n. Consider, for example, the node n with state $n.state = a$ and a list of active operators $n.active = \{b, c, d\}$. Assume, the state $\{ac\}$ can be pruned, because we can not reach a goal state from it. However, pruning the state $\{ac\}$, and thereby its subtree, is not the maximal pruning we can perform. The sibling $\{ab\}$ of $\{ac\}$ contains in its subtree a node with the state $\{abc\}$, a state from which we know already that we can not reach a goal state. Using the $n.active$ list, $OPUS$ avoids the generation of such states in step 8 of its algorithm. This type of pruning is still local, but includes the pruning information of the siblings.

The algorithm terminates, in the default settings, with only one goal state: the first that it finds. In order to allow for all goal states, small changes need to be done, see



```
Data structure:
Each node, $n$, in the search tree has associated with it three items of information:
$n.state$  the state from the search space that is associated with the node;
$n.active$  the set of operators to be explored in the sub-tree descending from the node; and
$n.mostRecentOperator$  the operator that was applied to the parent node's state to create the cur-
    rent node's state.
Algorithm:
 1. Initialize a list called $OPEN$ of unexpanded nodes as follows,
      (a) Set $OPEN$ to contain one node, the start node $s$.
      (b) Set $s.active$ to the set of all operators, $\{o_1, o_2, ...o_n\}$
      (c) Set $s.state$ to the start state.
 2. If $OPEN$ is empty, exit with failure; no goal state exists.
 3. Remove from $OPEN$ a node $n$, the next node to be expanded.
 4. Initialize to $n.active$ a set containing those operators that have yet to be examined, called
    $RemainingOperators$.
 5. Initialize to $\{\}$ a set of nodes, called $NewNodes$, that will contain the descendants of $n$ that
    are not pruned.
 6. Generate the children of $n$ by performing the following steps for every operator $o$ in $n.active$,
      (a) Generate $n'$, a node for which $n'.state$ is set to the state formed by application of $o$ to
          $n.state$.
      (b) If $n'.state$ is a goal state, exit successfully with the goal represented by $n'$.
      (c) Set $n'.mostRecentOperator$ to $o$.
      (d) Add $n'$ to $NewNodes$.
 7. While there is a node $n'$ in $NewNodes$ such that pruning rules determine that no sole re-
    maining goal state is accessible from $n'$ using only operators in $RemainingOperators$, prune
    all nodes in the search tree below $n'$ from the search tree below $n$ as follows,
      (a) Remove $n'$ from $NewNodes$.
      (b) Remove $n'.mostRecentOperator$ from $RemainingOperators$.
 8. Allocate the remaining operators to the remaining nodes by processing each node $n'$ in
    $NewNodes$ in turn as follows,
      (a) Remove $n'.mostRecentOperator$ from $RemainingOperators$.
      (b) Set $n'.active$ to $RemainingOperators$.
 9. Add the nodes in $NewNodes$ to $OPEN$.
10. Go to Step 2.
```

Figure 2. The $OPUS^o$ pseudo code, taken from Webb [1995]

[Webb, 1995]. Overall, $OPUS^s$ was designed for the search of nodes that satisfy some criteria. The $OPUS^o$ algorithm, instead, is an extension of the $OPUS^s$ algorithm, designed for optimization tasks. The algorithm requires two user defined functions: $value(n)$ and $optimisticValue(n, o)$. The $value(n)$ function assigns to each state of a node a corresponding value, and in a maximization task we are seeking for a node with a maximal value. The $optimisticValue(n, o)$ is supposed to answer whether we can find nodes with higher values in subbranches. This function is used to prune subbranches of a given node n and subbranches of its siblings. Additionally, it is used to prune nodes that are going to be visited next and for the heuristics used in $OPUS$. $OPUS^o$ coincides with the FSS algorithm, in that $OPUS^o$ assigns more active operators to nodes that have lower optimistic values. This heuristic ensures that the space placed under unpromising operators is maximized, and better pruning ratios may be achieved. The intuition is that nodes with a low optimistic value, including their operators, are more likely to get pruned away in an successive step. Similar to the $OPUS^s$ algorithm, by default, $OPUS^o$ only returns one solution, the node with the maximum value.

As mentioned, other approaches for unordered search had been developed. The approach of Schlimmer et al. [1993] ensures an optimal pruning of a fixed ordered search space. In order to achieve this, they store every pruned state. Their approach does not prevent the generation of states, from which we know already that they can be pruned. Moreover, it requires an exponential amount of memory. In contrast, $OPUS$ avoids the generation of states that can be pruned, using negligible storage overhead. The FSS algorithm ensures, similar to $OPUS$, that nodes, that have been identified as prunable, are not generated as the algorithm proceeds. The FSS algorithm uses a heuristic ordering of the operators in each node, such that the proportion of the search space placed under unpromising operators is maximized. $OPUS$ uses a similar approach for additional pruning. Using $OPUS$, the nodes that receive the maximum number of operators are the nodes with a small optimistic value. This is assumed to have similar effects to the heuristic ordering of FSS. Both FSS and $OPUS$ perform this operations using negligible computational overhead. Contrary to FSS, $OPUS$ eliminates the constraint that the values of the states can not increase, resulting of the application of additional operators. In contrast to the A^* algorithm, $OPUS$ dynamically organizes the search space in order to prune more parts of the space. Furthermore, it does not rely on the restriction, that the value of a state has to be equivalent to the sum of the costs of the operations leading to this state. The only requirements are that for at least one goal state g, which lies bellow any node n in the search tree, the optimistic value for node n is not smaller than the value of g, and the goal state has the largest value. These requirements, however, are reasonable for an optimistic function.

The complexity of the algorithm depends on how many states can be pruned away. Consider a node e with n active operators, if no operator can be pruned away the search space below e has a size of 2^n, and therefore $OPUS$ needs to explore 2^n states. Otherwise, if an operator can be pruned away, the search space below e reduces to a size of 2^{n-1}. Therefore, the search space below e gets approximately halved every time an operator gets pruned away. Generally, if p operators are removed, the size of the resulting search space equals to $\approx \frac{2^n}{2^p}$. In contrast to fixed order search, $OPUS$ prunes an exponentially growing amount of more nodes at each state.

$OPUS$ can also be applied to SGD. In order to do so, Webb [2001] adapted the algorithm to find impact rules, which is similar to SGD. Let P denote the global population and $y : P \rightarrow Y$, the function, that assigns each entity a target variable. Let $\mathcal{L}$ be the standard description language, that is, a conjunction of basic propositions. $OPUS$ creates the search space via applications of operators, which can be seen as a basic proposition and an application of two operators can be seen as a conjunction of two basic propositions. In detail, the algorithm starts with an initial state "true" and spans the search space via conjunctions of basic propositions. As outlined,

$OPUS$ requires a *value* function as input. One option is to use the impact measure $y(Q) = (mean(Q) - mean(P)) \times cover(Q)$, outlined in [Webb, 2001]; another option is to use the relative weighted accuracy, as outlined in the lecture. One important property of $OPUS$ is that it prunes parts of the search space using optimistic values (optimistic estimators). Considering this, Webb [2001] verifies after k rules have been recorded, if $\sum_{x \in cover(New):value(x) \geq mean(P)} (value(x) - mean(P)) < impact(IRule_n)$. If yes, $OPUS$ can terminate. By default, the $OPUS$ algorithm allows only for single solutions. Using the derivation in [Webb, 2001] allows for finding the k topmost sub groups according to the impact measure.

4 Evaluation

The $OPUS$ algorithm is general in its application. In the experimental part, however, Webb [1995] applies the algorithm to find pure conjunctive expressions that maximize a target function: the Laplace accuracy. The Laplace function measures how well a given conjunction (subgroup) covers a particular class. The datasets were taken from the UCI repository [Murphy and Aha, 1992]. These datasets consist of different classes, where each instance is described via a conjunction of attribute-value pairs. $OPUS$ aims to find for each class rules, that maximizes the Laplace function. In the experimental setup, $OPUS^o$ was used in different versions, see ([Webb, 1995], table 2). Somewhat confusingly, "no optimistic pruning", still applies optimistic pruning for the nodes in the list $OPEN$, which includes all the remaining nodes. However, as a comparison, the $OPUS$ algorithm is adapted to emulate a fixed order search. In order to emulate the fixed order search, $OPUS$ avoids the forward pruning for not yet generated states in the subtrees. This implies, that the fixed order emulation, by default, prunes only one state and the corresponding subtree, while $OPUS$ prunes the state and the subtree, plus additional nodes of the subtree of the siblings. This setup is not really fair, since algorithms outlined in Schlimmer et al. [1993] ensure an optimal pruning.

The number of nodes visited during the execution serves as a measure for goodness. It turns out that $OPUS$ outperforms fixed order search on every data set. This shows the power of the combination of the different, partly objective, pruning strategies. Therefore, it is not surprising that fixed order search can not handle spaces of size bigger than 2^{91}. In contrast, $OPUS$ is able to handle even the largest search space of size 2^{162}.

As mentioned, different versions of $OPUS$ where used. One of the versions is "no other pruning": this version is very similar to the fixed order search but includes the heuristic assignment of the operators to the child nodes. Anyway, the results show that the effect of the smart pruning using the lists of active operators does not have the effect as assumed. Throughout all data sets "no other pruning" is able to solve the search task using an acceptable amount of visited states.

Generally, the results show that the effect of the different parts of $OPUS$ depends on the data-set. The same holds for the decision, whether to use depth-first-search or best-first-search. In practice it may be suitable to try both. Summarizing, OPUS performs best using all of its components together on the data sets used in the experimental setup.

In an other work, Webb [2001] uses a variation of the $OPUS$ algorithm in order to search for impact rules. An impact rule consists of an *antecedent* and a *consequent*. The *antecedent* is a conjunction of basic propositions with respect to the data. The *consequent* is the set of numeric target variables. The goal is to find k impact rules (subgroups) which maximize the impact function, see Section 3. The $OPUS^{IR}$ algorithm is a sparse version of the $OPUS^o$ algorithm. It traverses the search space in a recursive depth-first search manner. Moreover, it cannot achieve the pruning rates of $OPUS^o$, since $OPUS^{IR}$ prunes similar to the fixed order search. In particular, it is interesting that $OPUS^o$ was not employed, since this version, as outlined, offers the best performance. One possible issue could be, that it was not possible to design an admissible heuristic. As it seems, it was important for the task at hand that no goal states were pruned away. In addition, no heuristics, such as the individual assignment of operators, are used. Summarizing, Webb [2001] shows that it is possible to find interesting rules. He compares his approach to one based on item set mining, which fails, however, because of memory issues.

[Webb, 1993] uses OPUS similarly, in order to learn classifiers to evaluate their precision (a classifier in this context is a set of rules). He showed that the heuristic maximization of Laplace, actually, leads to a decrease in classification performance. Webb [1994] uses $OPUS$ as an alternative approach for the induction of decision lists. In their theory of over searching, Quinlan and Cameron-Jones [1995] studied the problem of exhaustive search in comparison to heuristic search. They found out that an exhaustive search such as $OPUS$ often results in a lower classification performance. This problem occurs because the probability that a rule fits by chance is increased. Fürnkranz [1999] uses a derivation called MagnumOpus to find association rules.

5 Conclusion

The $OPUS$ algorithm was designed to allow for search tasks in very large search spaces. It utilizes pruning and optimistic pruning to systematically prune branches of the search space. Armed with an admissible heuristic for the $optimisticValues$, $OPUS$ ensures to return an optimal solution. There are many variations of the $OPUS$ algorithm. However, in this work, the author outlines two versions: the $OPUS^s$ and the $OPUS^o$ algorithm. The former is suitable for satisfying search, while the later is more suitable for optimization tasks. Generally, $OPUS$ needs to be adapted for the search at hand.

The different variations of the algorithm were already used in different domains such as machine learning and data mining. Among others, MagnumOpus was used to find associations rules, and $OPUS^o$ was used to find classifier that generalizes best a given set of data. The results show that $OPUS$ is able to efficiently search large spaces. On the other hand, specifically for the classification task it turns out that searching large parts of the space might be misleading in different contexts. Additionally, the results indicate, that pruning might lead to a longer search, since $OPUS$ might prune nearby goal states.

$OPUS^o$ requires two functions as a user defined input. These pruning rules, which are domain specific and therefore not defined in $OPUS$, can have important influence on the performance of the algorithm. If the functions are not chosen properly, $OPUS$ might prune away optimal goal states, or fail in finding an optimal state. Furthermore, it is unclear whether functions that do not solely rely on cover, such as functions based on information theory metrics, can be used in $OPUS$

Moreover, the results indicate that in a great deal of applications, for instance rule learning, it might be suboptimal to consider a lot of states. Furthermore, the question arises, whether it makes sense to consider the whole space, and whether it is possible to individualize the search of $OPUS$. Considering subgroup discovery, it might be tempting to search for large conjunction (or bounded by size, as in Webb [2001]) of basic proposition, since smaller descriptions might be more user friendly. Therefore, it might be better to use a different approach, because it is unclear whether we can adapt $OPUS$, and in particular the input pruning functions to the individual task at hand. This coincides with the observation that many practical applications do not make use of the $OPUS$ framework, even though nowadays the sizes of the search spaces are rather increasing. The $OPUS$ has the potential to act as a research tool for comparison. For example, to check the error of a heuristic search algorithm.

Considering the evaluation part, it would have been interesting to see how $OPUS$ performs against FSS and other common algorithms. Furthermore, a potential useful addition would be to see how $OPUS$ compares against A^* or the apriori algorithm.

The development of optimistic functions that depend on the $value(n)$ but are independent of how the $value(n)$ is defined, could be a potential interesting target of future research, that would allow $OPUS$ to generalize better to multiple domains.

References

Martin Atzmueller. Subgroup discovery. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 5(1):35–49, 2015.

Scott H Clearwater and Foster J Provost. Rl4: A tool for knowledge-based induction. In *Tools for Artificial Intelligence, 1990., Proceedings of the 2nd International IEEE Conference on*, pages 24–30. IEEE, 1990.

Johannes Fürnkranz. Separate-and-conquer rule learning. *Artificial Intelligence Review*, 13(1):3–54, 1999.

PM Murphy and DW Aha. Uci repository of machine learning databases. department of information and computer science, university of california, irvine, ca, 1992.

Patrenahalli M. Narendra and Keinosuke Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers*, 9(C-26):917–922, 1977.

Peter Norvig. *Paradigms of artificial intelligence programming: case studies in Common LISP*. Morgan Kaufmann, 1992.

Judea Pearl. Heuristics: intelligent search strategies for computer problem solving. 1984.

J Quinlan and R Cameron-Jones. Oversearching and layered search in empirical learning. *breast cancer*, 286:2–7, 1995.

Jeffrey C Schlimmer et al. Efficiently inducing determinations: A complete and systematic search algorithm that uses optimal pruning. In *ICML*, pages 284–290, 1993.

Jana Schmidt, Andreas Hapfelmeier, Marianne Mueller, Robert Perneczky, Alexander Kurz, Alexander Drzezga, and Stefan Kramer. Interpreting pet scans by structured patient data: a data mining case study in dementia research. *Knowledge and Information Systems*, 24(1):149–170, 2010.

Richard Segal and Oren Etzioni. Learning decision lists using homogeneous rules. In *AAAI*, pages 619–625, 1994.

Geoffrey I Webb. *Systematic search for categorical attribute-value data-driven machine learning*. Deakin University, School of Computing and Mathematics, 1993.

Geoffrey I Webb. Recent progress in learning decision lists by prepending inferred rules. In *SPICIS94: Proceedings of the Second Singapore International Conference on Intelligent Systems*, pages B280–B285, 1994.

Geoffrey I Webb. Opus: An efficient admissible algorithm for unordered search. *Journal of Artificial Intelligence Research*, 3:431–465, 1995.

Geoffrey I Webb. Discovering associations with numeric variables. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 383–388. ACM, 2001.