Anytime Inductive Logic Programming

Tony Lindgren

Department of Computer and Systems Sciences
Stockholm University and Royal Institute of Technology
Electrum 230, 164 40 Kista, Sweden
tony@dsv.su.se

Abstract

[1] Anytime algorithms refers to algorithms that "always" can produce a result, often the result of the algorithm depends on the time at hand, the longer the time the better the answer. In this thesis we present an easy way of turning regular Inductive Logic Programming (ILP) algorithms such as Divide-And-Conquer (DAC) and Separate-And-Conquer (SAC) into anytime algorithms. We also introduce key-concepts of ILP and give a brief history of the field. We also conduct experiments with these anytime algorithms and introduce a simple heuristic called *squared quota*, that we compare with an established one, *information gain*. It seems that *squared quota* is better suited for small *windows* of example data, and hence better to use in anytime systems. A comparison between SAC and DAC reveals that they excel in different combinations of examples/background knowledge.

[1]This thesis corresponds to 20 weeks of full-time work.

Contents

1 Introduction

This thesis is aiming at four different objectives. The first aim is that it could be read and understood by an audience with little or no knowledge about Inductive Logic Programming (ILP). The second objective is to give a short history of ILP, and hence better the odds for success of objective one. The third objective is to look at how established learning strategies in ILP can be converted to anytime learning strategies. The fourth objective is to do experiments with these new strategies and to discuss the implications of the experiment.

In section 2 we will try to give an overview of ILP. The overview includes a short history of the area, and displays some of its key issues. In section 3 we will describe the algorithms that we are going to scrutinize, and the setting of our experiment. In section 4 we will give a summary of the results, discuss the results and the implications of them.

2 Overview of ILP

In this section we will give an overview of the history of ILP and some of the key issues that separate different ILP strategies from each other.

The area of artificial intelligence (AI) that studies learning is called *machine learning*. According to [6] there are four main approaches to *machine learning*, these are *decision trees*, *neural networks*, *genetic algorithms* and *Inductive Logic Programming*. ILP uses different algorithms to create hypotheses with respect to training data, and possibly background knowledge. By a logical program is meant a program written in a language that is based on logic [1]. An example of such a programming language is Prolog [8], which is the dominating language in this genre. One can regard an ILP system as a system which from positive and negative examples, i.e. it's training data, creates rules that cover the positive examples and exclude the negative ones. By covering we mean that the (logic) program, after being exposed to training data, should be able to classify them correctly, i.e. to accept positive examples, and reject negative. These learned hypothesis that we get after exposing our ILP system to a training set can be used for classifying other examples than those in the training set. We can then get a measure on the quality of the rules induced by classifying a set of examples reserved for testing. A perfect strategy would need few learning examples and would classify all of the testing examples correctly. In reality this is not the case.

2.1 Key issues of ILP

Here we will give a brief overview of some key issues of different ILP concepts. For a more thorough treatment of these issues the reader is recommended to consult [5] or [6, pages 163 – 174].

Inductive Logic Programming is the process of getting from a set of examples to a theory that support these examples, possibly with respect to the background knowledge. The background knowledge defines the terms in which we may express the theory. This is an example adopted and slightly changed from [5]. Suppose our background knowledge about fishes is:

$$B = \begin{cases} canSwim(X) \leftarrow fish(X) \\ livesInWater(X) \leftarrow fish(X) \\ fish(X) \leftarrow salmon(X) \end{cases}$$

Suppose also that our Example set contains:

$$E^+ = \begin{cases} canSwim(malin) \leftarrow \\ livesInWater(malin) \leftarrow \end{cases}$$

Then it would be feasible for us to induce the following hypothesis:

$$H = fish(malin) \leftarrow$$

This is correct according to our background information and from the information in example set. However this can of course be a faulty assumption if we know that malin is an dolphin, which is not a fish. This shows the importance of a "complete" set of examples.

Also with the above example it would be correct to induce:

$$H = salmon(malin) \leftarrow$$

This however intuitively "feels" more speculative than inducing that malin is a fish. This issue of justification of induced hypotheses have been argued among philosophers and no consensus has yet emerged. Based on the fact that salmon is a subset of fish I'm in the opinion that it is less speculative to induce fish than salmon, based on probability.

2.1.1 Normal problem setting

The Normal problem setting is the most common setting of ILP systems. It is either in a *definitive* setting or in a *general* setting. The *definitive* setting is a subclass of the *general* setting, and hence also simpler. This is the case since the *definitive setting* refers to using *SLD-refutations* [2] instead of *SLDNF-refutations* [3] as is the case of the *general* setting. Very simplified one can say that the difference between *SLD-refutations* and *SLDNF-refutations* is that the latter

[2]'SLD' stands for 'SL-resolution for Definite clauses', and 'SL' for 'Linear resolution with a selection function' [6]

[3]'SLDNF' stands for 'SLD-resolution with negation as failure'

allows the use of negation, which the former does not. The type of negation that is used is not really what we ordinary mean by negation in logic. If we for instance would try to satisfy the statement:

$\neg salmon(malin) \leftarrow$

With regard to the background information above and the induced hypothesis, this would be unsatisfiable because malin is a salmon results in *True*, according to the hypothesis, then we negate that and get *False*, hence it is unsatisfied. But if we try to satisfy:

$\neg shark(malin) \leftarrow$

With regard to the background information above, this would be satisfiable, why? Well the case for this is that SLDNF-refutations uses a notion of *Closed World Assumption*, *CWA*. In our example it would amount to that we first check if there exists an clause $shark(malin) \leftarrow$ in the background knowledge or in the hypothesis we induced, and if it does not, (as in our example) we get *False*, which we negate to get *True*. So one can say that if that we don't find a clause in our "world" then we by *CWA* concludes that we give it a *False* value that we later negate, which becomes *True*. Prolog internally uses *SLDNF-refutations* when it tries to satisfy it's goals.

After we have exposed an ILP system to examples and learning is done, then the learned hypotheses together with the background knowledge should imply all the positive examples (for completeness) and not imply any of the negative examples (for consistency).

2.1.2 Nonmonotonic problem setting

The main difference to the normal problem setting is that in a *nonmonotonic* setting it does not exists any explicit examples. Instead the positive examples are said to implicitly be part of the background knowledge. And the negative examples are derived from the background knowledge, by a kind of *Closed World Assumption*. The hypothesis that we get from an ILP-system in a nonmonotonic setting differs from that what we get from an system that are in normal setting, in that the latter *imply* the positive examples, but in the former the hypothesis should be *True* with respect to the background information, and hence also the positive examples, which where derived from the background knowledge. Because the nonmonotonic setting does not have explicit examples it can derive more restrictive hypothesis than in the normal problem setting. The nonmonotonic setting is often used in *data mining*, which often is used in databases to find regularities.

2.1.3 Top-down and Bottom-up learning

Depending on what form of learning an ILP system performs it is either a Top-down system or a Bottom-up system. The difference between the two approaches is that in a Top-down system, the system starts with an *overly general* hypothesis that it specializes. With specialize we mean the process of converting a hypothesis to another hypothesis such that the first hypothesis is more general with regard to the second hypothesis and to a *generality order* on clauses. Examples of a *generality order* is subsumption and logical implication. For definitions of subsumption and logical implication see [1], or any introduction book to logic. An *overly general* clause is characterized by that all the positive examples are covered and a subset of the negative examples are covered as well. An example adopted from [3] might clear things up.

Our overly general clause is:

```
reward(S,R):- suit(S), rank(R).
```

Background knowledge:

```
suit(S) :- red(S).
suit(S) :- black(S).
rank(R) :- num(R).
rank(R) :- face(R).
red(hearts).
red(diamonds).
black(spades).
black(clubs).
num(1).  ...  num(10).
face(jack).
face(queen).
face(king).
```

Positive and negative examples:

$E^{+} = \{$ `reward(spades,7), reward(clubs,3)` $\}$

$E^{-} = \{$ `reward(hearts,5), reward(clubs,jack)` $\}$

As we see here our overly general theory covers all of the examples, both positive and negative. A specialization of the theory could be, according to our background knowledge, `reward(S,R) :- red(S), rank(R).`, because `reward(S,R) :- suit(S), rank(R).` subsumes `reward(S,R) :- red(S), rank(R).` with regard to the background knowledge. This is perfectly legal but the clause `reward(S,R) :- red(S), rank(R).` does not really help us much in finding a good theory given our example set as it does not cover any positive examples. It is up to the learning strategy to address this problem, we will later come back to this example when we describe the algorithms that are to be scrutinized.

The direct opposite of Top-down learning is Bottom-up learning. Instead of starting with an overly general theory Bottom-up learning starts with an *overly* *specific* theory. This means that the theory covers none of the negative examples but does not cover all of the positive examples. Instead of taking a specialization step as in the case of Top-down learning we now do a *generalization* step which manifests itself in a theory that is more general according to a *generality order* with regard to the first clause. An example of this could be an the overly specific clause, `reward(clubs,R) :- num(R).`. This clause covers none of the negative examples but only one of the positive examples. A generalization step could be the creation of a clause, `reward(S,R) :- black(S), num(R).`. This theory is exactly what we are looking for, it covers all of the positive examples but none of the negative. It should be noted that some ILP learning system uses both specialization and generalization to achieve learning, but always have a Top-down or Bottom-up strategy, that can be "reversed" in some special situations.

2.1.4 Other differences of ILP systems

In incremental learning the system is fed with one example at a time, it does not know the contents of the examples that it yet haven't seen. A batch learning system is a system that always have access to all the available examples.

Another difference among systems is whether they are interactive with the user, which then is called an *Oracle* and can answer questions put forward by the ILP system. This gives the learning system additional information. The non-interactive system does not have an *Oracle* to ask and can hence only use the information that is derivable from the examples.

Yet another issue is the use of bias in the ILP system. Bias puts different forms of constraints on a ILP system. According to [6] there are tree different kinds of bias: language, search and validation bias.

- The language bias regulates which theories that are to be accepted by the system, and thus narrows the systems need to handle clauses other than that of a specific type.
- The search bias is concerned with the way a system searches through permitted clauses and theories.
- The validation bias determines when a learned theory is acceptable and when learning will stop.

2.2 A Brief History of ILP

This section rely heavily on [7] and [6, pages 174–177], otherwise indicated.

Philosophers first studied the notion of induction. Aristotle characterized science roughly as deduction from first principles, which where to be obtained by induction from experience. Later in history the notion of induction popped up from time to time in philosophy. One of the philosophers that addressed the notion of induction was David Hume who formulated the "problem of induction" or "Hume's problem". The problem Hume addressed was: how can induction from a finite number of cases result in knowledge about the infinity of cases to which an induced rule applies? What justifies inferring a general rule from a finite number of cases? The answer to the last questions is according to Hume is that there are no justification. It is a fact that humans exposed to some pattern without irregularities, we expect that that pattern should repeat itself in a similar context that we seen it before.

In the 20th century those who was interested in induction were (again) philosophers and mathematicians who developed formal logic. Their interest in induction was often in the *probability* that a theory or hypothesis receives from empirical data.

In 1956 came a book named *A study of thinking*, which was written by Bruner, Goodnow and Austin and dealt with the information processing aspects of the brain. They made experiment with the human ability to categorize observations. One of their findings was that when humans looked at different concepts they had the ability to apply that concept to other similar concepts, i.e. induction. They also notice three different strategies for learning, based on the type of concept to be learned. The concepts was of the types: conjunctive, disjunctive and relational. These findings would later be used in ILP.

In the same decade another area of science recognized induction, namely Artificial Intelligence (AI). The interest that AI had in induction was that instead of putting all the necessary information into systems when they were constructed, and perhaps miss some valuable information, one could construct a system with a minimum of information and let it adapt and learn from its experience. These was (and still is) the ambition but nobody really knows how to realize it.

In the early 1960's in the area of pattern recognition, they tried to find the best way to represent patterns. This lead to the development of symbolic description languages. Banerji showed a language based on predicate logic for pattern recognition, which had much more expressive power than other pattern recognition languages at the time. One of the advantages of this work was the ability of the language to grow, i.e. to use previously learned concepts as a basis for later concepts.

In 1970 Plotkin developed the notion of *relative least general generalization* which is an extension of *least general generalization*. This finding had great influence on other researchers in the same area. During 1970 Ryszard Michalski made an attempt to build a learning system based on Logic programming called INDUCE which made some important contributions of how to make the system's concept descriptions more readable for humans and he also noticed the importance of *background knowledge*. Steven Vere continued the work of Plotkin and developed a technique that was able to recognize association chains. He also developed the notion of *counterfactuals* which really are exception to concepts. Brian Choen developed a incremental learning system called CONFUCIUS which was the first system to be able to reuse learned concepts.

In the early 1980's we saw the first systems where the learned concepts could be executed as a logical program. These systems are the first systems that can be said to perform inductive logic programming. One of the systems was MIS developed by Ehud Shapiro which showed important concepts as a *contradiction backtracking algorithm*, which could check if an theory was too general i.e. it implied an negative example and a *refinement* operator which could be used to specialize a theory.

From the middle 1980's up till now (1999), the interest in issues that later would become ILP increased all the time, much because of the growing interest in logic programming and Prolog. In the second half of 1980, Buntine generalized the notion of subsumption relative to the background theory, to make subsumption more usable. During the same time period Muggleton together with Buntine introduced the notion of *inverse resolution*, and used that notion as an operator in their system CIGOL, for making generalization steps and for predicate invention. Other alternatives to *inverse resolution* was given by Quinlan which was implemented in his Foil system which used information based heuristics and refined its Horn clauses in an top-down fashion. Muggleton and Feng used Plotkin's *relative least generalization* in their bottom-up system GOLEM.

In 1990, Muggleton coined the expression inductive logic programming, and defined this field as the intersection between logic programming and machine learning. In 1991 the first International Workshop on ILP was organized. Since then the workshop have been repeated every year. The area of ILP is today an area of great activity and interest, with a lot of different issues that are open to exploration.

2.3 Uses of ILP

ILP system have been put to use in a lot of different areas. One of their advantages over other approaches of machine learning is that the generated hypothesis often is more readable to humans. Another advantage is the use of background knowledge in the learning process which can be used to describe relations among entities. The only real downside contra other systems is the relative inefficiency of today's ILP systems. We will now describe some cases of the practical uses of ILP systems, all of these cases are described more thoroughly in [2].

Case 1: Finite element mesh design
Finite element mesh design concerns computing the physical stress in some object. The method uses a mesh that is so constructed that the object is divided in small sections by the mesh and each section has a certain behavior. When the engineers go about separating objects with a mesh, there aren't any tools available that helps the engineer in doing so. And if a mesh is to fine grained where it should not be it leads to more computational cost than necessary. And even worse if it's too coarse where it should not be it leads to errors in the calculations. Using a positive example set consisting of objects with a "good"

mesh, the ILP system GOLEM was able to create hypothesis that help engineers in the area to “see” interesting relational dependencies.

Case 2: Predicting the mutagenicity of chemical compounds
The Structure/Activity Relationship (SAR) of chemical compounds is often used in the design of drugs. When the ILP system PROGOL was used to predict mutagenicity on a data set that earlier had been predicted by regression analysis and neural networks, the result of PROGOL was as good as the other systems. But as the hypotheses of an ILP system are easier for humans to understand than those of other ML approaches, the hypothesis gave new chemical insights because a new structural feature that is an alert for mutagenicity was found.

Case 3: Biological classification of river water quality
By observing various biological species present in water, one can establish the quality of it. When applied on data derived from rivers in the British Midlands, the ILP systems GOLEM and CLAUDIEN were able to create hypothesis that according to experts in the field seemed to be right and very appealing due to their explicitness. A neural network was also applied to the same data, but its results where not as easy to understand as that of the ILP systems.

3 Converting ILP to anytime ILP

Instead of letting the algorithms completely learn from their training set, we will give the different algorithms a certain amount of time to their disposal, and we will then look at their classifying ability. For this scheme to work we have to give our learning strategies an 'anytime algorithm' behavior. By that we mean that they 'always' can give a hypothesis to a requesting entity.

In our experiments we implement the 'anytime algorithm' behavior by writing every induced rule to a certain file. From that file we can in a multitasking environment ask questions against that file, and let the learning process keep on going all along. We will only have to read in the file at certain intervals so that we always have the latest version of the rules.

3.1 Separate-and-Conquer

Separate-and-Conquer searches the hypothesis space and tries to find a rule that covers only positive examples and no negative examples. When it finds such a rule it adds it to its hypothesis, hence its overall hypothesis will be a conjunction of the hypothesis that it has found. It then removes the covered positive examples and starts to search for new rules. When no positive examples are left to cover it finishes. Here we use resolution as a specialization operator. More formally:
O is our overly general clause, B is our Background knowledge and H is our conjunction of hypothesis

while $E^+ \neq \{\}$ do
    if O not covers any $e \in E^-$ according to B then
        add O to hypothesis H
        and remove from E^+ the examples covered by O
        continue with the algorithm with the original O clause
    if O covers some $e \in E^-$ according to B then
        resolve upon O creating a more specialized clause that covers
        some $e \in E^+$ according to B
        continue the algorithm with this clause as O

As promised in section 2.1.3 we will now use the background knowledge, positive and negative examples described in that section to give an example of how SAC would go about to generate a hypothesis. Here we assume that we have a heuristic measure that always selects the best resolvent. First SAC would evaluate the condition in the while loop to be true as:

$E^+ = \{$ `reward(spades,7), reward(clubs,3)` $\}$

Then it would check the first if statement and evaluate that to false as:

$E^- = \{$ `reward(hearts,5), reward(clubs,jack)` $\}$

Our overly general clause `reward(S,R) :- suit(S), rank(R).` covers both of the negative examples according to the background knowledge. SAC then considers the last if statement which becomes true and now we resolve our overly general clause to `reward(S,R) :- black(S), rank(R).`. As mentioned before it's up to the heuristic measure to decide which literal to resolve upon, and which of the resolvents to choose. As we take another turn in our algorithm the while loop evaluates to true, as E^+ is not empty. The first-if statement will be false as we cover one of the negative examples `reward(clubs,jack).` Now our last if-statement evaluates to true and we resolve upon `reward(S,R) :- black(S), rank(R).` which becomes `reward(S,R) :- black(S), num(R).` The while loop evaluates to true, the first if-statement also evaluates to true as we don't cover any negative example. We then add `reward(S,R) :- black(S), num(R).` to our hypothesis, and remove the covered positive examples which leaves us with the empty set. We then continue with our original overly general clause `reward(S,R) :- suit(S), rank(R).`. This time however the while loop evaluates to false and hence we are finished.

3.2 Divide-and-Conquer

Divide-and-Conquer is quite similar to SAC but instead of getting *one* clause of a specialization operator DAC gets *all* possible specializations with regard to the

literal to be unfolded. It also behaves different when it has found a hypothesis that it adds to its conjunction of hypothesis, in that it does not 'begin again' from the top and instead uses the last clause in its queue of clauses. Here we use unfolding as a specialization operator. More formally:
O is our overly general clause, B is our Background knowledge and H is our conjunction of hypothesis

if O not covers some $e \in E^-$ according to B then
    add O to hypothesis H
    and remove from E^+ the examples covered by O
if O not covers some $e \in E^+$ according to B then
    do nothing
if O covers some $e \in E^-$ according to B then
    unfold upon O creating more specialized clauses of O called $R_1, \cdots, R_n$
    Let for all $1 \leq i \leq n$
    $E^+_{R_i}$ be subset of E^+ covered by a resolvent R_i
    continue the algorithm with clause R_i and $E^+_{R_i}$

If we consider DAC with our example we will find that the first if-statement will become false as we cover both of the negative examples. DAC then considers the second if-statement which also becomes false as we cover both of the positive examples. The last if-statement however becomes true so we resolve upon `reward(S,R) :- suit(S), rank(R).` which becomes $R_1 =$ `reward(S,R) :- red(S), rank(R).` and $R_2 =$ `reward(S,R) :- black(S), rank(R).` if we unfold on `suit(S)`, as mentioned before a heuristic decides which literal to resolve upon. Now we start the algorithm again with $R_1 =$ `reward(S,R) :- red(S), rank(R).` and this ends abruptly as the clause does not cover any positive examples. With $R_2 =$ `reward(S,R) :- black(S), rank(R).` we now start the algorithm over again. The first evaluates to false as we cover one negative examples. The second if-statement evaluates to false. The last if-statement however evaluates to true so we resolve upon $R_2 =$ `reward(S,R) :- black(S), rank(R).` to get $R_{2_1} =$ `reward(S,R) :- black(S), num(R).` and $R_{2_2} =$ `reward(S,R) :- black(S), face(R).`. Now we start the algorithm over again. The first if-statement evaluates to false. The second evaluates to true as we cover no negative examples, we then add to our $R_{2_1} =$ `reward(S,R) :- black(S), num(R).` conjunction of hypothesis H. Next we remove all E^+ covered by $E^+_{R_{21}}$ which leaves us with the empty set. $R_{2_2} =$ `reward(S,R) :- black(S), face(R).` evaluates to false on the first if-statement, but the second if-statement evaluates to true and we are finished.

3.3 Experimental setting

The algorithms will have a certain amount of time to their disposal and when time is consumed no more rules can be induced. All experiments where conducted on an Sun SPARCstation 4 running Solaris operating system. All algorithms where coded in SICStus 3 #6 prolog. The time at disposals for the algorithms where 0.1s, 0.5s, 1.0s, 5.0s, 10.0s, 50.0s, 100.0s, 500.0s, 1000.0s. All sessions where compiled, and the learning was stopped with the time_out predicate at the specified times. The examples were randomly divided in two equal sized halves so that one half could be used for training and the other half for testing. The aim of the experiments is to evaluate the performance of the different algorithms with regard to the time factor. For instance is SAC more suitable in anytime learning than DAC? Which heuristic is best suited for anytime learning? In this experiment we test two different ILP approaches against each other: SAC and DAC, described in section 3.1 and 3.2. We use three different types of heuristic in our experiments; *random*, *squared quota* and *information gain*, the latter two are described below. Random heuristic needs no explanation. In our experiments we also uses different *windows*, which means for example if we have a window of 20 then 10 negative examples and 10 positive examples will be the source of information given to the heuristic to choose which to resolve upon. The windows in our experiment are 20, 40, 60, 80, 100, 120. The properties we measure are; correctness, number of rules produced, and whether or not the algorithm finished. For a description of the datasets used see appendix A, for the results of the experiments see appendix B and in appendix C figures of the result is shown.

3.4 Heuristic used

As mentioned in section 3.1 and 3.2 it is up to the heuristic used to choose which of the literal in the body to resolve upon and which of the resolvents to continue with (SAC) alternatively which path to follow first (DAC).

3.4.1 Squared quota

squared quota $= \frac{cov(R,E^+)^2}{cov(R,E^-)+1}$

Where $cov(R, E)$ denotes the number of examples E that is covered by R. Squared quota works as follows: The case for which we use the squared quota instead of just quota is because if we have two clauses such that clause one covers 100 positive examples and 10 negative examples, which gives us a quota of 10, and clause two covers 1000 positive examples and 100 negative we also get 10 as quota. If one reasons a little bit one sees that the second clause probably is more promising to resolve upon. Using squared quota the quota for clause one would almost be 909 and for the second clause 9900, which gives it a higher rating than that of clause one. The 1 is there so that we can't get zero in the denominator.

3.4.2 Information gain

information gain $= cov(R, E^+) \left(log_2 \frac{cov(R,E^+)}{cov(R,E^+ \cup E^-)} - log_2 \frac{cov(C,E^+)}{cov(C,E^+ \cup E^-)} \right)$
Where $cov(R, E)$ denotes the number of examples E that is covered by R. Information gain works as follows: The entropy of a resolvent R of C, is measured with $log_2 \frac{cov(R,E^+)}{cov(R,E^+ \cup E^-)}$ that is, it measures the diversity of the examples covered by R. If R only covers positive examples then the entropy is zero. So if we have a clause C that has the entropy 1, then it covers as many negative as positive examples. Then the resolvent R of C that gets the lowest entropy is the best resolvent.

3.4.3 Implementation of heuristic

The way that the above heuristics are implemented in the SAC and DAC is as follows: In SAC the employment of the heuristics is done by that all the literals in the body is unfolded upon, so that we get all possible clauses that we could specialize from our original overly general clause. The heuristic is applied to each of the clauses and the clause that get the highest value is used to continue our algorithm with. When these heuristics are applied to DAC the same approach is used to select a clause to continue with but with the difference that we also add all the other resolvents of the same clause as a queue to be processed after we are finished with the best resolved clause. The reader should note that the author have used an unorthodox version of information gain with the DAC approach.

4 Discussion

See appendix B for a presentation of the results that are subject to discussion.

4.1 Summary of results

4.1.1 Results for Tic-Tac-Toe

- SAC performs better than DAC in accuracy and produces less rules.
- Squared quota has higher accuracy than information gain and produces less rules in both SAC and DAC, it is also faster than information gain.

4.1.2 Results for Tic-Tac-Toe with intermediate predicate

- SAC performs better than DAC in accuracy and produces less rules, it is also faster.
- Squared and information gain have almost the same accuracy, but squared quota produces less rules in both SAC and DAC.

4.1.3 Results for Shuttle

- SAC and DAC are equal in accuracy but SAC produces less rules.
- Squared and information gain has the same accuracy.

4.1.4 Results for Promoters

- SAC have lower accuracy than DAC but produces less rules.
- Squared and information gain have the same accuracy.

4.1.5 Results for King+Rook versus King+Pawn on a7

- SAC have lower accuracy than DAC but produces less rules.
- Squared quota have higher accuracy than information and produces fewer rules.

4.2 DAC versus SAC

In our experiments DAC have better accuracy than SAC in two domains, is equal to SAC in one domain and has lower accuracy in two domains. Therfore if one only considers accuracy it's a dead race. But if one considers the number of rules produced, then SAC is better or equal in all domains. In an comparison of their speed, both approaches have almost similar performance. Note though that our implementation of the DAC heuristic is a little bit ad hoc [4] which gives it a worse performance than it should be with an better implementation. One interesting thing to note in the experiment is that SAC is faster than DAC in Tic-Tac-Toe with intermediate predicates which is surprising. Otherwise one can't see any big differences other than that DAC produces more rules in a shorter time as it doesn't need to "start again" all the way from the top. This is often a problem though, referred to as the replication problem see [4]. The differences inherited in how the algorithm induces rules mean that they are suited to different datasets. For example: in the Tic-Tac-Toe dataset SAC is both as fast as DAC but has higher accuracy and fewer rules. In the Promoters domain the DAC approach still produces more rules, but now it has higher accuracy

[4] See section 5.1 Further topics

than the SAC approach. If one look at the Shuttle domain they perform almost identically.

4.3 Squared quota versus information gain

The results of the experiments reveal that squared quota has a small advantage over information gain, in accuracy, speed and in number of rules produced.

4.4 Window size

The issue of window size and of how it affects the performance of the algorithms is like choosing which algorithm to use, it depends on the domain. In some domains the accuracy doesn't suffer so badly by having a small window size but often more rules are produced. As the window size increase the number of rules produced often shrinks this is most obvious using information gain heuristics. So generally a small window size produces more rules, faster and possibly with less accuracy than a big window size.

5 Concluding remarks

In this thesis we have given a short overview of the history of ILP, and have presented the key issues in the field. We have also shown how one can convert regular ILP approaches to anytime ILP approaches, which could be of a benefit to time critical systems. We also introduced a new heuristic called squared quota, which seems to perform better, or at least as well, as information gain. In the comparison of SAC and DAC as to their suitability as anytime learner there is no real winner, both perform with almost similar speed in some cases though one of them is a little bit faster than the other. In accuracy the story is the same in a particular domain one approach can be better than the other but as a whole they perform equally good. SAC is better in two of five domains so is also DAC and in the fifth domain they have similar performance. The only difference that is constant is that DAC always produces more or at least equally many rules as SAC.

5.1 Further topics

As the case is now the window of examples that is fed to the ILP algorithm is always the same. One improvement could be to make the selection of examples completely random. Of course this would create more overhead.

One improvement that could be added to DAC is the use the information gain as:
$-c\sum_{i=1}^{n} cov(C_i,E^+)log_2\frac{cov(C_i,E^+)}{cov(C_i,E^+\cup E^-)} + cov(C_i,E^-)log_2\frac{cov(C_i,E^-)}{cov(C_i,E^+\cup E^-)}$
The C_i with the lowest value is the best literal for DAC to use as resolvent.

$cov(C,E)$ denotes the number of elements of E that are covered by C and c is a constant $1/cov(C, E^+ \cup E^-)$.

A similar scheme for squared quota would look like:
$\sum_{i=1}^{n} \frac{cov(C_i,E^+)^2}{cov(C_i,E^-)+1} + \frac{cov(C_i,E^-)^2}{cov(C_i,E^+)+1}$
where again $cov(C,E)$ denotes the number of elements of E that are covered by C. These two heuristics selects the best literal to resolve upon then we use our heuristics described in section 3.4 to select the most promising clause to explore first. At this point another improvement would be to sort queue of clauses according to the heuristic.

One interesting thing to look more into is also the heuristic of squared quota and how well it stands to more established heuristics, especially with a more complete set of examples than was the case in the experiments in this thesis.

More advanced types of anytime ILP strategies can also be considered; for example it may well be possible for the learning algorithm to adapt it's learning parameters, i.e. window size, heuristic, strategy, with regard to previous experience. In this case we obviously mean a situated learner within a certain context, which in itself is changeable. If one follows the line of thought above one might imagine some type of embodied system that uses anytime ILP learning for long term and maybe short term behavior, possibly together with other means of decision making.

References

[1] M. Ben-Ari. *Mathematical Logic for Computer Science.* Prentice Hall, 1993.

[2] Ivan Bratko and Stephen Muggleton. Applications of inductive logic programming. *Communications of the ACM*, 1995.

[3] Boström H. Covering vs. divide-and-conquer for top-down induction of logic programs. *Proc. of the Fourteenth International Joint Conference on Artificial Intelligence*, 1995.

[4] Boström H. and Asker L. Combining divide-and-conquer and separate-and-conquer for efficient and effective rule induction. *Proceedings of the Ninth International Workshop on Inductive Logic Programming*, 1999.

[5] Stephen Muggleton and Luc de Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 1994.

[6] Shan-Hwei Nienhuys-Cheng and Ronald de Wolf. *Foundations of Inductive Logic Programming.* Springer-Verlag, 1997.

[7] Claude Sammut. The origins of inductive logic programming: A prehistoric tale. In *The Third International Workshop on Inductive Logic Programming ILP'93*, 1993.

[8] Robert W. Sebesta. *Concepts of Programming Languages*. Addison-Wesley, 1996.

A Example domains

A.1 Tic-Tac-Toe

The tic-tac-toe domain was taken from the UCI Machine Learning Repository, and was encoded in Prolog by Henrik Boström. The background knowledge for this domain was:

```
win_for_x(S1,S2,S3,S4,S5,S6,S7,S8,S9):-
    square(S1),square(S2),square(S3),
    square(S4),square(S5),square(S6),
    square(S7),square(S8),square(S9).
square(b).
square(o).
square(x).
```

The total number of examples were 960, where 627 (65.3%) was positive examples and 333 (34.7%) where negative examples. So the total number of training examples were 480. This database encodes the complete set of possible board configurations at the end of tic-tac-toe games, where "x" is assumed to have played first. The target concept is "win for x" (i.e., true when "x" has one of 8 possible ways to create a "three-in-a-row"). An alternative representation background knowledge where also subject to experiment:

```
win_for_x(S1,S2,S3,S4,S5,S6,S7,S8,S9):-
    square(S1),square(S2),square(S3),
    square(S4),square(S5),square(S6),
    square(S7),square(S8),square(S9).
square(S):- o_or_b(S).
square(x).
o_or_b(b).
o_or_b(o).
```

The intention is that as the background information is more general, one will be able to learn rules faster that have higher accuracy. See [3] for additional results without anytime behavior.

A.2 Shuttle

The shuttle domain was taken from the UCI Machine Learning Repository, and was encoded in Prolog by Henrik Boström. The background knowledge for this domain was:

```
auto(Stability,Error,Sign,Wind,Magnitude,Visibility):-
```

```
    stability(Stability),
    error(Error),
    sign(Sign),
    wind(Wind),
    magnitude(Magnitude),
    visibility(Visibility).
stability(stab).
stability(xstab).
error(xl_error).
error(lx_error).
error(mm_error).
error(ss_error).
sign(pp_sign).
sign(nn_sign).
wind(head_wind).
wind(tail_wind).
magnitude(low_magnitude).
magnitude(medium_magnitude).
magnitude(strong_magnitude).
magnitude(out_of_range_magnitude).
visibility(visibility).
visibility(novisibility).
```

The total number of examples were 278, where 133 (47.8%) was positive examples and 145 (52.2%) where negative examples. So the total number of training examples were 139. This database describes weather or not an auto landing module should or should not be activated during the landing of a space shuttle.

A.3 Promoters

The Promoters domain was taken from the UCI Machine Learning Repository, and was encoded in Prolog by Malin Hendrenius. The background knowledge for this domain was:

```
promoters(A1, ... ,A57):-
    nucleotide(A1),
    ...
    nucleotide(A57).
nucleotide(X):- purine(X).
nucleotide(X):- pyrimidine(X).
purine(a).
purine(g).
pyrimidine(t).
pyrimidine(c).
```

The total number of examples were 106, where 53 (50.0%) was positive examples and 53 (50.0%) where negative examples. So the total number of training examples were 53. The database contains examples that can be used to predict the member/non-membership of class of sequences with biological promoter activity (promoters initiate the process of gene expression).

A.4 King+Rook versus King+Pawn on a7

The Promoters domain was taken from the UCI Machine Learning Repository, and was encoded in Prolog by Henrik Boström. The background knowledge for this domain was:

```
win(X1, ... ,X36):-
    boolean(X1),
    ...
    boolean(X12),
    a13(X13),
    boolean(X14),
    a15(X15),
    boolean(X16),
    ...
    boolean(X35),
    a36(X36).
boolean(t).
boolean(f).
a13(l).
a13(g).
a15(n).
a15(w).
a15(b).
a36(n).
a36(t).
```

The total number of examples were 3196, where 1669 (52.2%) was positive examples and 1527 (47.8%) where negative examples. So the total number of training examples were 1598. This database holds examples for how King+Rook will prevent the queening of a pawn currently on square a7.

B Results

In the figures below there are nine columns Time, Measure, SAC/DAC and SAC-X/DAC-X 20, ... , SAC-X/DAC-X 120. Time is the time that the algorithms have at their disposal, measured in seconds. Measure describes the three performance measures measured they are, Finished which refers to whether or not the algorithm was finished. T stands for Time out, that the algorithm didn't finish learning and was prematurely interrupted. S stands for, success, which means that it finished learning. Another measure is Num. Cl. which stands for the number of clauses that our hypothesis have, and finally Accuracy refers to how well it classified the test set in %. The SAC/DAC column describes the results from an SAC/DAC with random heuristic. The SAC-X/DAC-X 20, ... , SACH-X/DACH-X 120 where X is substituted for squared quota or information gain. The specified heuristic considers a "window" of 20 to 120 examples, that is total set of examples, so a window of 20 gives 10 positive examples and 10 negative examples.

B.1 Results Tic-Tac-Toe

Time	Measure	SAC	SAC-squared quota					
			20	40	60	80	100	120
0.1	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
0.5	Finished	T	T	T	T	T	T	T
	Num. Cl.	2	1	0	0	0	0	0
	Accuracy	34.65	42.80	0	0	0	0	0
1.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	4	1	1	1	0	0	0
	Accuracy	40.08	42.80	42.80	42.80	0	0	0
5.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	27	9	7	5	4	3	3
	Accuracy	70.35	54.49	64.72	62.63	63.47	59.50	55.32
10.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	77	20	15	11	8	7	6
	Accuracy	94.15	67.01	84.97	79.54	79.33	74.53	65.97
50.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	82	39	23	22	20	23	23
	Accuracy	95.82	96.24	98.75	97.91	99.16	97.08	99.16
100.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	97	39	23	22	20	23	23
	Accuracy	97.49	96.24	98.75	97.91	99.16	97.08	99.16
500.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	76	39	23	22	20	23	23
	Accuracy	97.70	96.24	98.75	97.91	99.16	97.08	99.16
1000.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	138	39	23	22	20	23	23
	Accuracy	95.40	96.24	98.75	97.91	99.16	97.08	99.16

Time	Measure		SAC-information gain					
			20	40	60	80	100	120
0.1	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
0.5	Finished	T	T	T	T	T	T	T
	Num. Cl.	2	0	0	0	0	0	0
	Accuracy	34.65	0	0	0	0	0	0
1.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	4	0	0	0	0	0	0
	Accuracy	40.08	0	0	0	0	0	0
5.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	27	1	1	1	1	1	1
	Accuracy	70.35	35.70	35.70	36.95	36.95	36.95	42.80
10.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	77	2	2	1	1	2	1
	Accuracy	94.15	35.91	36.33	36.95	36.95	41.95	42.80
50.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	82	13	10	8	7	8	9
	Accuracy	95.82	38.62	39.87	38.20	41.54	50.10	69.10
100.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	97	28	21	16	15	17	20
	Accuracy	97.49	45.72	42.59	39.46	42.59	65.14	78.29
500.0	Finished	S	S	T	T	S	S	S
	Num. Cl.	76	148	111	85	78	52	51
	Accuracy	97.70	78.91	76.62	78.08	95.20	94.78	97.08
1000.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	138	148	146	125	78	52	51
	Accuracy	95.40	78.91	84.34	89.56	95.20	94.78	97.08

Time	Measure	DAC	DAC-squared quota					
			20	40	60	80	100	120
0.1	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
0.5	Finished	T	T	T	T	T	T	T
	Num. Cl.	21	1	0	0	0	0	0
	Accuracy	40.71	42.80	0	0	0	0	0
1.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	46	7	4	1	1	0	0
	Accuracy	50.52	44.05	43.42	42.80	42.80	0	0
5.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	176	59	33	23	17	13	11
	Accuracy	77.04	62.00	51.36	48.64	48.43	46.14	44.68
10.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	168	113	62	48	37	27	21
	Accuracy	79.54	79.75	78.29	72.03	70.56	54.49	49.90
50.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	185	113	79	67	62	85	84
	Accuracy	78.29	79.75	88.94	93.53	97.08	89.98	89.98
100.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	169	113	79	67	62	85	84
	Accuracy	77.66	79.75	88.94	93.53	97.08	89.98	89.98
500.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	168	113	79	67	62	85	84
	Accuracy	72.44	79.75	88.94	93.53	97.08	89.98	89.98
1000.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	160	113	79	67	62	85	84
	Accuracy	77.04	79.75	88.94	93.53	97.08	89.98	89.98

Time	Measure	DAC	DAC-information gain					
			20	40	60	80	100	120
0.1	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
0.5	Finished	T	T	T	T	T	T	T
	Num. Cl.	21	1	0	0	0	0	0
	Accuracy	40.71	35.70	0	0	0	0	0
1.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	46	11	3	2	0	0	0
	Accuracy	50.52	39.46	37.37	41.96	0	0	0
5.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	176	67	36	20	16	12	10
	Accuracy	77.04	56.16	48.85	44.05	47.18	44.67	44.67
10.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	168	134	82	49	34	25	22
	Accuracy	79.54	67.64	71.19	53.03	51.98	50.10	51.15
50.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	185	188	156	141	136	127	126
	Accuracy	78.29	76.20	81.21	82.67	83.92	83.09	85.39
100.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	169	188	156	141	136	127	126
	Accuracy	77.66	76.20	81.21	82.67	83.92	83.09	85.39
500.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	168	188	156	141	136	127	126
	Accuracy	72.44	76.20	81.21	82.67	83.92	83.09	85.39
1000.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	160	188	156	141	136	127	126
	Accuracy	77.04	76.20	81.21	82.67	83.92	83.09	85.39

B.2 Results for Tic-Tac-Toe with intermediate predicate

Time	Measure	SAC	SAC-squared quota					
			20	40	60	80	100	120
0.1	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
0.5	Finished	T	T	T	T	T	T	T
	Num. Cl.	1	1	1	0	0	0	0
	Accuracy	37.36	42.80	42.80	0	0	0	0
1.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	2	2	3	1	1	1	1
	Accuracy	35.07	51.36	51.36	42.80	42.80	42.80	42.80
5.0	Finished	T	S	S	S	T	T	T
	Num. Cl.	10	13	17	8	8	7	6
	Accuracy	39.46	100.0	100.0	100.0	91.65	83.92	83.92
10.0	Finished	T	S	S	S	S	S	S
	Num. Cl.	23	13	17	8	9	9	8
	Accuracy	48.43	100.0	100.0	100.0	100.0	100.0	100.0
50.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	154	13	17	8	9	9	8
	Accuracy	89.14	100.0	100.0	100.0	100.0	100.0	100.00
100.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	174	13	17	8	9	9	8
	Accuracy	89.77	100.0	100.0	100.0	100.0	100.0	100.00
500.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	162	13	17	8	9	9	8
	Accuracy	90.19	100.0	100.0	100.0	100.0	100.0	100.00
1000.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	158	13	17	8	9	9	8
	Accuracy	88.52	100.0	100.0	100.0	100.0	100.0	100.00

Time	Measure	SAC	SAC-information gain					
			20	40	60	80	100	120
0.1	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
0.5	Finished	T	T	T	T	T	T	T
	Num. Cl.	1	1	0	0	0	0	0
	Accuracy	37.36	36.53	0	0	0	0	0
1.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	2	2	1	1	1	1	1
	Accuracy	35.07	37.58	35.49	34.66	35.70	35.70	42.80
5.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	10	12	9	7	6	5	4
	Accuracy	39.46	48.85	46.56	36.95	40.92	45.09	51.98
10.0	Finished	T	T	T	T	T	T	S
	Num. Cl.	23	28	20	14	12	10	11
	Accuracy	48.43	65.76	60.13	43.01	49.27	47.81	100.0
50.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	154	55	55	57	44	30	11
	Accuracy	89.14	100.0	100.0	100.0	100.0	100.0	100.00
100.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	174	55	55	57	44	30	11
	Accuracy	89.77	100.0	100.0	100.0	100.0	100.0	100.00
500.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	162	55	55	57	44	30	11
	Accuracy	90.19	100.0	100.0	100.0	100.0	100.0	100.00
1000.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	158	55	55	57	44	30	11
	Accuracy	88.52	100.0	100.0	100.0	100.0	100.0	100.00

Time	Measure	DAC	DAC-squared quota					
			20	40	60	80	100	120
0.1	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
0.5	Finished	T	T	T	T	T	T	T
	Num. Cl.	4	0	0	0	0	0	0
	Accuracy	35.70	0	0	0	0	0	0
1.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	20	3	1	1	0	0	0
	Accuracy	41.34	45.72	42.80	42.80	0	0	0
5.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	120	42	23	15	10	10	7
	Accuracy	90.34	57.83	50.10	46.76	49.68	45.72	46.35
10.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	131	82	50	37	29	22	17
	Accuracy	86.01	72.86	61.38	56.16	58.04	56.78	55.74
50.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	109	101	78	77	70	66	74
	Accuracy	89.56	88.52	89.77	90.40	95.20	95.41	94.78
100.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	127	101	78	77	70	66	74
	Accuracy	84.34	88.52	89.77	90.40	95.20	95.41	94.78
500.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	100	101	78	77	70	66	74
	Accuracy	89.14	88.52	89.77	90.40	95.20	95.41	94.78
1000.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	105	101	78	77	70	66	74
	Accuracy	86.22	88.52	89.77	90.40	95.20	95.41	94.78

Time	Measure	DAC	DAC-information gain					
			20	40	60	80	100	120
0.1	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
0.5	Finished	T	T	T	T	T	T	T
	Num. Cl.	4	0	0	0	0	0	0
	Accuracy	35.70	0	0	0	0	0	0
1.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	20	3	1	0	0	0	0
	Accuracy	41.34	36.95	36.53	0	0	0	0
5.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	120	41	24	17	12	9	7
	Accuracy	90.34	46.76	44.68	44.89	48.43	47.60	45.93
10.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	131	76	52	37	28	19	15
	Accuracy	86.01	64.93	49.48	52.82	54.28	51.77	55.32
50.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	109	121	116	101	90	95	74
	Accuracy	89.56	82.25	83.09	86.01	88.31	90.61	96.45
100.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	127	121	116	101	90	95	74
	Accuracy	84.34	82.25	83.09	86.01	88.31	90.61	96.45
500.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	100	121	116	101	90	95	74
	Accuracy	89.14	82.25	83.09	86.01	88.31	90.61	96.45
1000.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	105	121	116	101	90	95	74
	Accuracy	86.22	82.25	83.09	86.01	88.31	90.61	96.45

B.3 Results Shuttle

Time	Measure	SAC	SAC-squared quota					
			20	40	60	80	100	120
0.1	Finished	T	T	T	T	T	T	T
	Num. Cl.	1	0	0	0	0	0	0
	Accuracy	59.71	0	0	0	0	0	0
0.5	Finished	S	S	T	T	T	T	T
	Num. Cl.	13	5	2	1	1	1	1
	Accuracy	99.28	94.96	92.09	86.33	86.33	86.33	74.82
1.0	Finished	S	S	S	T	T	T	T
	Num. Cl.	3	5	5	4	3	2	2
	Accuracy	75.54	94.96	99.28	97.84	96.40	93.53	89.21
5.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	11	5	5	5	5	5	5
	Accuracy	93.53	94.96	99.28	99.28	99.28	99.28	94.96
10.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	15	5	5	5	5	5	5
	Accuracy	94.24	94.96	99.28	99.28	99.28	99.28	94.96
50.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	21	5	5	5	5	5	5
	Accuracy	98.56	94.96	99.28	99.28	99.28	99.28	94.96
100.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	10	5	5	5	5	5	5
	Accuracy	82.73	94.96	99.28	99.28	99.28	99.28	94.96
500.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	10	5	5	5	5	5	5
	Accuracy	89.21	94.96	99.28	99.28	99.28	99.28	94.96
1000.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	19	5	5	5	5	5	5
	Accuracy	98.56	94.96	99.28	99.28	99.28	99.28	94.96

Time	Measure	SAC	SAC-information gain					
			20	40	60	80	100	120
0.1	Finished	T	T	T	T	T	T	T
	Num. Cl.	1	0	0	0	0	0	0
	Accuracy	59.71	0	0	0	0	0	0
0.5	Finished	S	T	T	T	T	T	T
	Num. Cl.	13	1	2	1	1	1	0
	Accuracy	99.28	58.27	92.09	86.33	86.33	86.33	0
1.0	Finished	S	T	S	T	T	T	T
	Num. Cl.	3	4	5	3	2	2	1
	Accuracy	75.54	60.43	99.28	96.40	93.53	93.53	74.82
5.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	11	21	5	5	5	5	5
	Accuracy	93.53	90.65	99.28	99.28	99.28	99.28	94.96
10.0	Finished	S	T	S	S	S	S	S
	Num. Cl.	15	21	5	5	5	5	5
	Accuracy	94.24	90.65	99.28	99.28	99.28	99.28	94.96
50.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	21	21	5	5	5	5	5
	Accuracy	98.56	90.65	99.28	99.28	99.28	99.28	94.96
100.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	10	21	5	5	5	5	5
	Accuracy	82.73	90.65	99.28	99.28	99.28	99.28	94.96
500.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	10	21	5	5	5	5	5
	Accuracy	89.21	90.65	99.28	99.28	99.28	99.28	94.96
1000.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	19	21	5	5	5	5	5
	Accuracy	98.56	90.65	99.28	99.28	99.28	99.28	94.96

Time	Measure	DAC	DAC-squared quota					
			20	40	60	80	100	120
0.1	Finished	T	T	T	T	T	T	T
	Num. Cl.	2	0	0	0	0	0	0
	Accuracy	56.83	0	0	0	0	0	0
0.5	Finished	T	S	S	T	T	T	T
	Num. Cl.	13	5	5	6	2	1	1
	Accuracy	79.86	94.96	97.84	99.28	93.53	86.33	74.82
1.0	Finished	T	S	S	S	S	S	S
	Num. Cl.	40	5	5	6	6	6	5
	Accuracy	89.93	94.96	97.84	99.28	99.28	99.28	94.96
5.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	36	5	5	6	6	6	5
	Accuracy	87.77	94.96	97.84	99.28	99.28	99.28	94.96
10.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	33	5	5	6	6	6	5
	Accuracy	87.77	94.96	97.84	99.28	99.28	99.28	94.96
50.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	29	5	5	6	6	6	5
	Accuracy	86.33	94.96	97.84	99.28	99.28	99.28	94.96
100.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	27	5	5	6	6	6	5
	Accuracy	95.68	94.96	97.84	99.28	99.28	99.28	94.96
500.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	28	5	5	6	6	6	5
	Accuracy	79.86	94.96	97.84	99.28	99.28	99.28	94.96
1000.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	20	5	5	6	6	6	5
	Accuracy	90.64	94.96	97.84	99.28	99.28	99.28	94.96

Time	Measure	DAC	DAC-information gain					
			20	40	60	80	100	120
0.1	Finished	T	T	T	T	T	T	T
	Num. Cl.	2	0	0	0	0	0	0
	Accuracy	56.83	0	0	0	0	0	0
0.5	Finished	T	T	S	T	T	T	T
	Num. Cl.	13	2	5	3	1	1	0
	Accuracy	79.86	92.09	97.84	93.53	86.33	86.33	0
1.0	Finished	T	S	S	S	S	S	S
	Num. Cl.	40	5	5	6	6	6	5
	Accuracy	89.93	99.28	97.84	99.28	99.28	99.28	94.96
5.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	36	5	5	6	6	6	5
	Accuracy	87.77	99.28	97.84	99.28	99.28	99.28	94.96
10.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	33	5	5	6	6	6	5
	Accuracy	87.77	99.28	97.84	99.28	99.28	99.28	94.96
50.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	29	5	5	6	6	6	5
	Accuracy	86.33	99.28	97.84	99.28	99.28	99.28	94.96
100.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	27	5	5	6	6	6	5
	Accuracy	95.68	99.28	97.84	99.28	99.28	99.28	94.96
500.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	28	5	5	6	6	6	5
	Accuracy	79.86	99.28	97.84	99.28	99.28	99.28	94.96
1000.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	20	5	5	6	6	6	5
	Accuracy	90.64	99.28	97.84	99.28	99.28	99.28	94.96

B.4 Results Promoters

Time	Measure	SAC	SAC-squared quota					
			20	40	60	80	100	120
0.1	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
0.5	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
1.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	2	0	0	0	0	0	0
	Accuracy	52.83	0	0	0	0	0	0
5.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	13	0	0	0	0	0	0
	Accuracy	67.92	0	0	0	0	0	0
10.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	23	0	0	0	0	0	0
	Accuracy	75.47	0	0	0	0	0	0
50.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	22	1	0	0	0	0	0
	Accuracy	86.79	50.94	0	0	0	0	0
100.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	21	2	1	1	1	1	1
	Accuracy	77.36	58.49	64.15	62.26	62.26	62.26	62.26
500.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	18	4	4	4	4	4	4
	Accuracy	86.79	64.15	73.58	71.70	71.70	71.70	71.70
1000.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	19	4	4	4	4	4	4
	Accuracy	77.36	64.15	73.58	71.70	71.70	71.70	71.70

Time	Measure	SAC	SAC-information gain					
			20	40	60	80	100	120
0.1	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
0.5	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
1.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	2	0	0	0	0	0	0
	Accuracy	52.83	0	0	0	0	0	0
5.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	13	0	0	0	0	0	0
	Accuracy	67.92	0	0	0	0	0	0
10.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	23	0	0	0	0	0	0
	Accuracy	75.47	0	0	0	0	0	0
50.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	22	1	1	0	0	0	0
	Accuracy	86.79	47.17	56.60	0	0	0	0
100.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	21	2	2	1	1	1	1
	Accuracy	77.36	47.17	58.49	62.26	62.26	62.26	62.26
500.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	18	10	4	4	4	4	4
	Accuracy	86.79	73.58	71.70	71.70	71.70	71.70	71.70
1000.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	19	10	4	4	4	4	4
	Accuracy	77.36	73.58	71.70	71.70	71.70	71.70	71.70

Time	Measure	DAC	DAC-squared quota					
			20	40	60	80	100	120
0.1	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
0.5	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
1.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	3	0	0	0	0	0	0
	Accuracy	54.72	0	0	0	0	0	0
5.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	20	0	0	0	0	0	0
	Accuracy	81.13	0	0	0	0	0	0
10.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	20	0	0	0	0	0	0
	Accuracy	69.81	0	0	0	0	0	0
50.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	15	3	0	0	0	0	0
	Accuracy	81.13	54.72	0	0	0	0	0
100.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	23	9	6	5	1	5	1
	Accuracy	83.02	73.58	77.36	77.36	62.26	77.36	62.26
500.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	23	10	8	7	4	7	4
	Accuracy	71.70	79.25	86.79	86.79	71.70	86.79	71.70
1000.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	18	10	8	7	4	7	4
	Accuracy	69.81	79.25	86.79	86.79	71.70	86.79	71.70

Time	Measure	DAC	DAC-information gain					
			20	40	60	80	100	120
0.1	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
0.5	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
1.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	3	0	0	0	0	0	0
	Accuracy	54.72	0	0	0	0	0	0
5.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	20	0	0	0	0	0	0
	Accuracy	81.13	0	0	0	0	0	0
10.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	20	0	0	0	0	0	0
	Accuracy	69.81	0	0	0	0	0	0
50.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	15	6	2	0	0	0	0
	Accuracy	81.13	58.49	67.92	0	0	0	0
100.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	23	11	7	6	6	6	6
	Accuracy	83.02	60.38	84.91	79.25	79.25	79.25	79.25
500.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	23	21	8	8	8	8	8
	Accuracy	71.70	81.13	86.79	86.79	86.79	86.79	86.79
1000.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	18	21	8	8	8	8	8
	Accuracy	69.81	81.13	86.79	86.79	86.79	86.79	86.79

B.5 Results King+Rook versus King+Pawn on a7

Time	Measure	SAC	SAC-squared quota					
			20	40	60	80	100	120
0.1	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
0.5	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
1.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
5.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	1	1	1	1	0	0	0
	Accuracy	47.87	66.15	66.15	66.15	0	0	0
10.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	2	1	1	1	1	1	1
	Accuracy	47.87	66.15	66.15	66.15	66.15	66.15	66.15
50.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	11	3	4	3	2	2	2
	Accuracy	50.69	67.52	93.55	86.05	85.98	85.98	85.98
100.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	23	6	6	4	4	3	3
	Accuracy	51.38	72.28	93.55	86.30	91.11	91.30	91.30
500.0	Finished	T	S	S	T	S	T	T
	Num. Cl.	195	82	20	20	23	14	17
	Accuracy	79.10	97.81	99.06	97.25	99.00	94.49	98.44
1000.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	138	82	20	29	23	25	21
	Accuracy	69.15	97.81	99.06	99.12	99.00	99.31	99.06

Time	Measure	SAC	SAC-information gain					
			20	40	60	80	100	120
0.1	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
0.5	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
1.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
5.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	1	1	1	1	0	0	0
	Accuracy	47.87	66.15	66.15	66.15	0	0	0
10.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	2	1	1	1	1	1	1
	Accuracy	47.87	66.15	66.15	66.15	66.15	66.15	66.15
50.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	11	4	4	3	3	3	2
	Accuracy	50.69	66.33	91.24	86.17	91.11	91.11	85.98
100.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	23	9	8	5	5	4	4
	Accuracy	51.38	67.52	91.36	86.42	91.11	91.24	86.36
500.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	195	44	37	26	24	19	15
	Accuracy	79.10	73.15	93.74	91.56	95.31	94.12	91.86
1000.0	Finished	S	T	S	T	S	T	S
	Num. Cl.	138	91	77	55	51	41	32
	Accuracy	69.15	79.72	97.75	94.93	98.25	97.25	95.37

Time	Measure	DAC	DAC-squared quota					
			20	40	60	80	100	120
0.1	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
0.5	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
1.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
5.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	13	1	1	1	1	0	0
	Accuracy	48.25	66.15	66.15	66.15	66.15	0	0
10.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	37	1	1	1	1	1	1
	Accuracy	52.75	66.15	66.15	66.15	66.15	66.15	66.15
50.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	260	17	15	16	14	3	2
	Accuracy	69.52	80.79	90.43	92.24	90.49	87.98	82.67
100.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	255	64	23	24	22	18	18
	Accuracy	90.49	89.92	93.80	93.87	92.30	92.24	92.24
500.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	365	82	26	26	27	29	29
	Accuracy	88.36	97.81	99.24	99.25	99.19	98.87	99.44
1000.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	361	82	26	26	27	29	29
	Accuracy	87.80	97.81	99.24	99.25	99.19	98.87	99.44

Time	Measure	DAC	DAC-information gain					
			20	40	60	80	100	120
0.1	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
0.5	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
1.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	0	0	0	0	0	0	0
	Accuracy	0	0	0	0	0	0	0
5.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	13	1	1	1	0	0	0
	Accuracy	48.25	66.15	66.15	66.15	0	0	0
10.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	37	1	2	1	1	1	1
	Accuracy	52.75	66.15	82.67	66.15	66.15	66.15	66.15
50.0	Finished	T	T	T	T	T	T	T
	Num. Cl.	260	55	21	13	10	3	3
	Accuracy	69.52	72.47	92.30	85.73	90.55	87.23	83.04
100.0	Finished	S	T	T	T	T	T	T
	Num. Cl.	255	105	36	38	23	20	17
	Accuracy	90.49	75.28	92.62	90.55	92.30	92.30	85.54
500.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	365	191	51	77	46	53	68
	Accuracy	88.36	94.99	98.06	97.93	98.25	97.98	98.87
1000.0	Finished	S	S	S	S	S	S	S
	Num. Cl.	361	191	51	77	46	53	68
	Accuracy	87.80	94.99	98.06	97.93	98.25	97.98	98.87

C Figures

Here the x-axis denotes time in seconds and the y-axis accuracy in percent. The time scale is different on the different figures depending on the slowest algorithms finishing time.

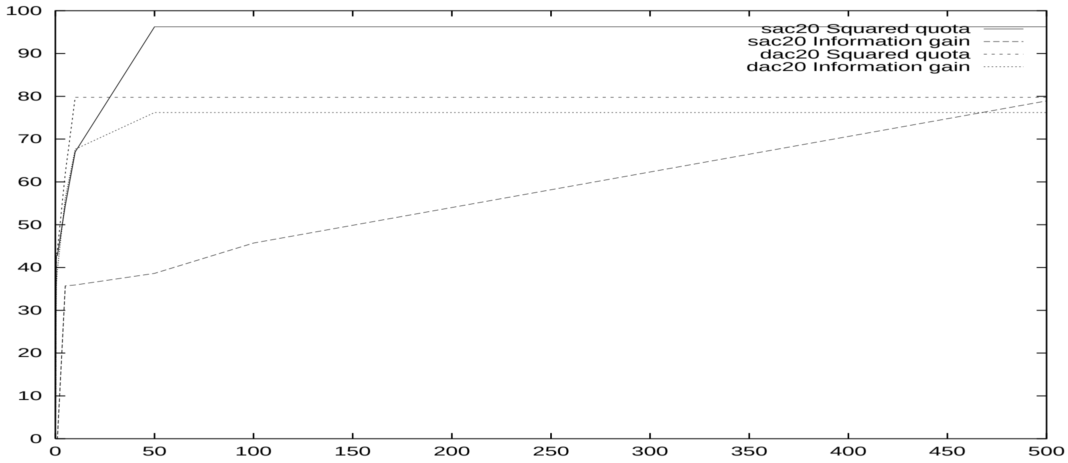

Figure 1: Accuracy in the Tic-Tac-Toe domain with windowsize 20.

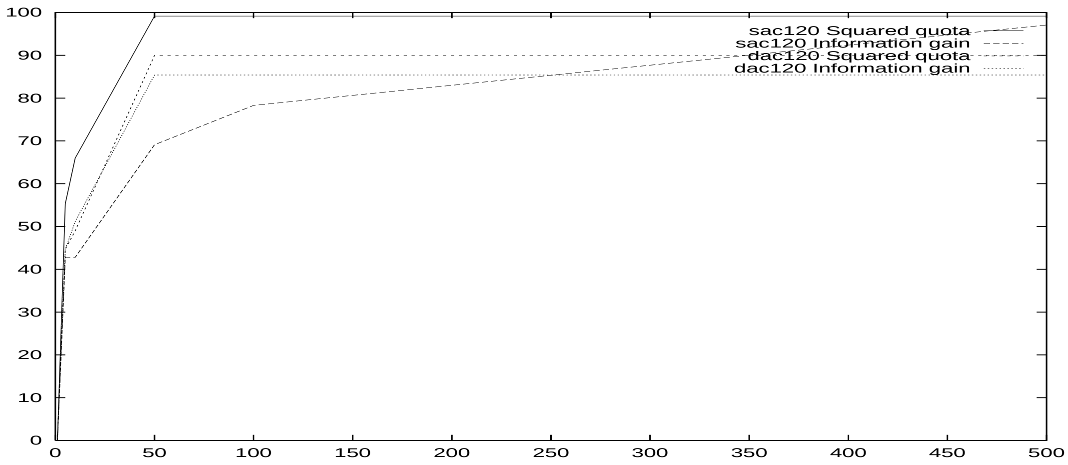

Figure 2: Accuracy in the Tic-Tac-Toe domain with windowsize 120.

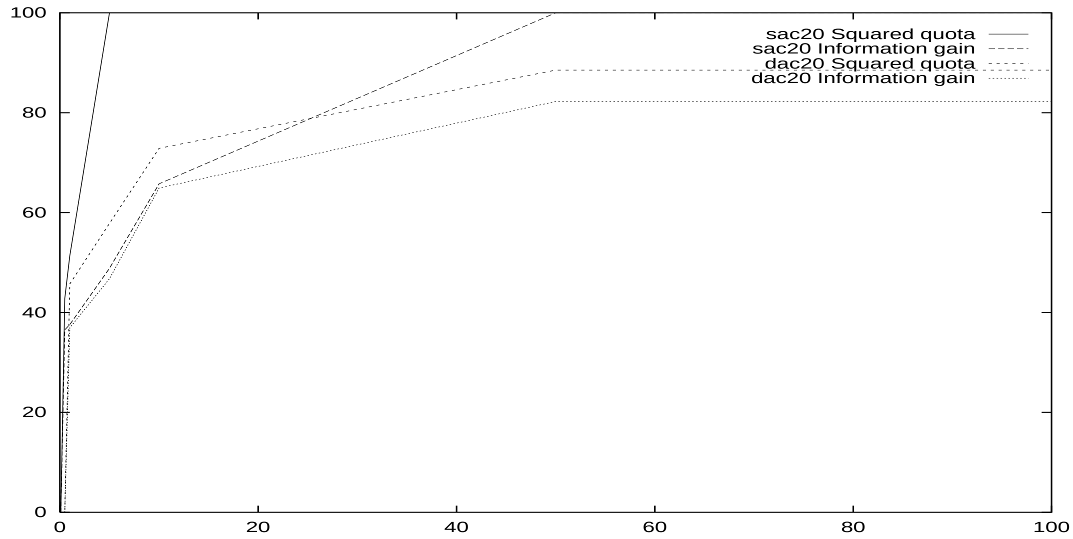

Figure 3: Accuracy in the int. Tic-Tac-Toe domain with windowsize 20.

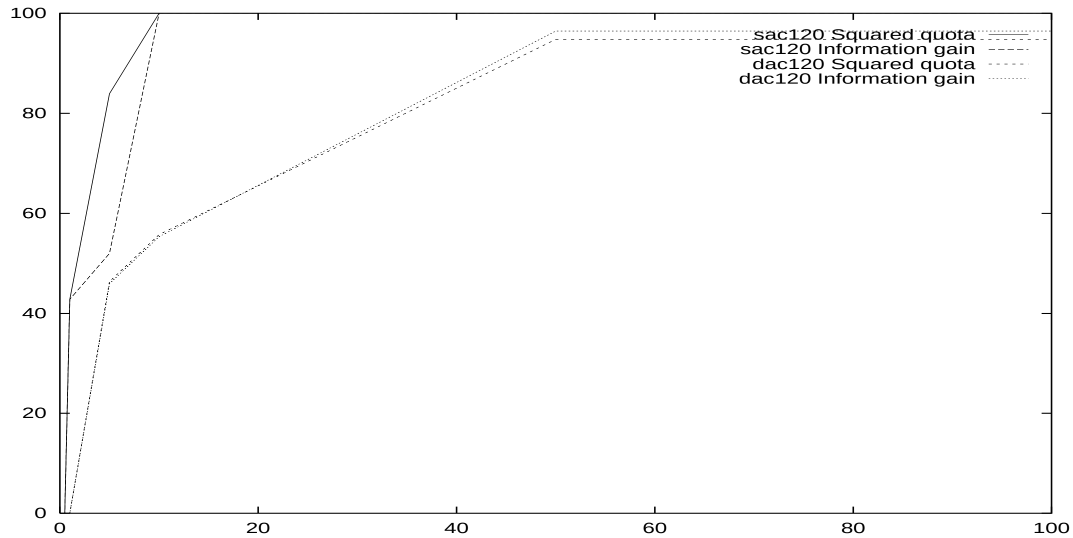

Figure 4: Accuracy in the int. Tic-Tac-Toe domain with windowsize 120.

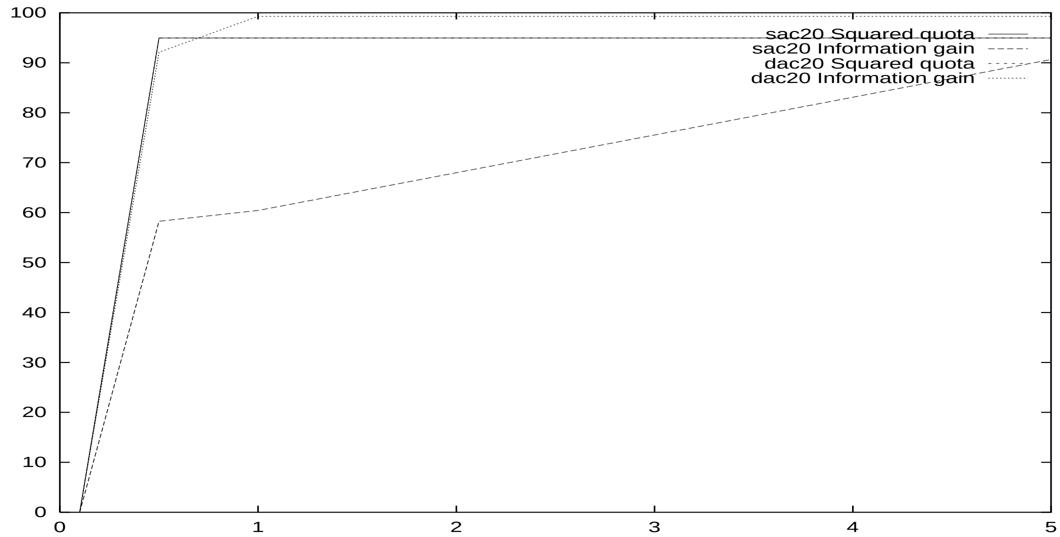

Figure 5: Accuracy in the shuttle domain with windowsize 20.

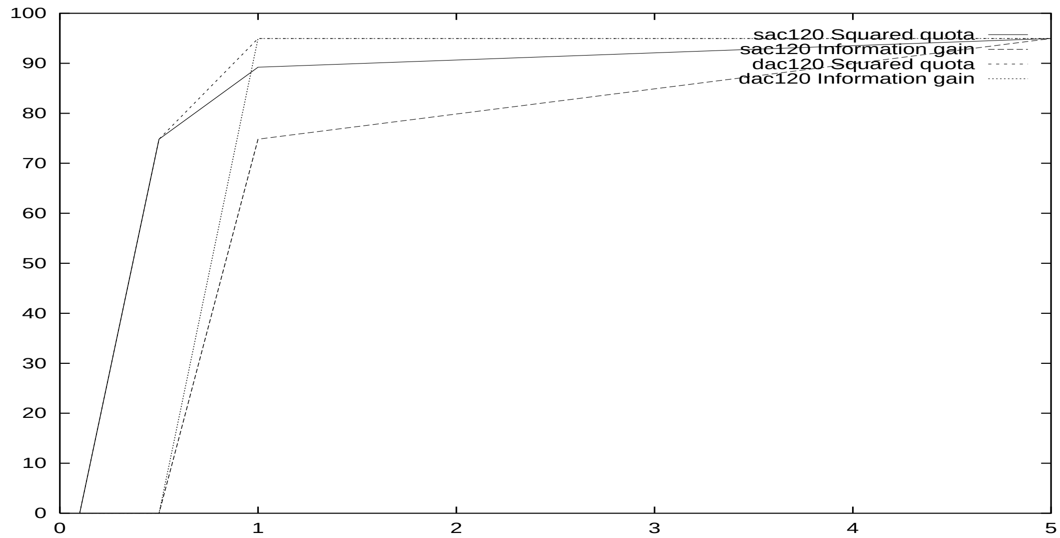

Figure 6: Accuracy in the shuttle domain with windowsize 120.

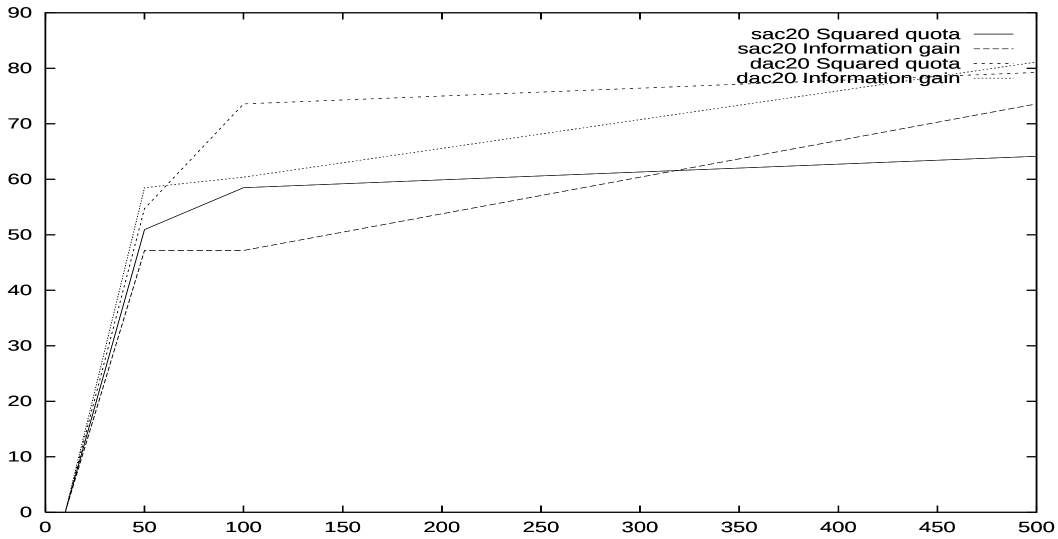

Figure 7: Accuracy in the promoters domain with windowsize 20.

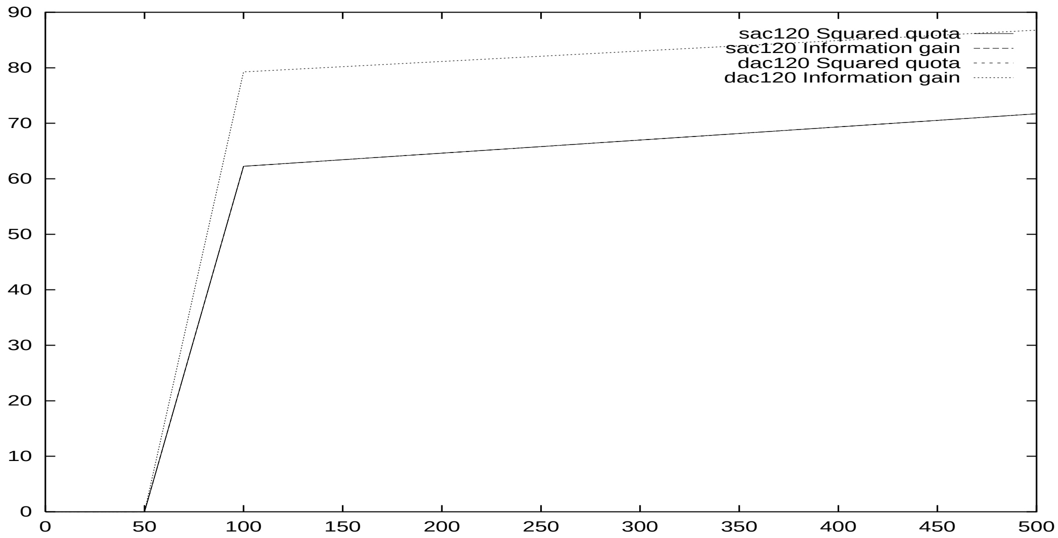

Figure 8: Accuracy in the promoters domain with windowsize 120.

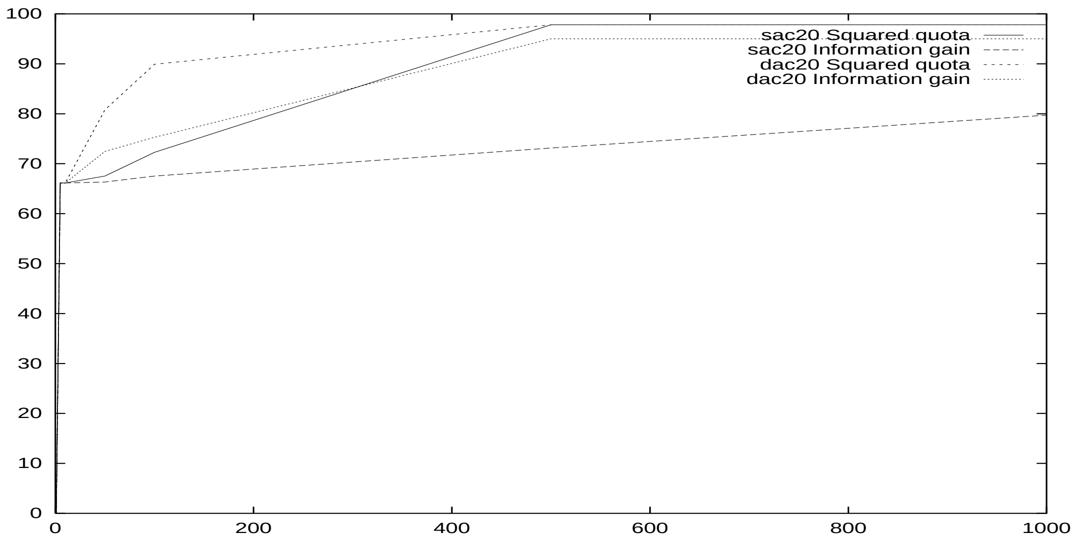

Figure 9: Accuracy in the King+Rook versus King+Pawn on a7 domain with windowsize 20.

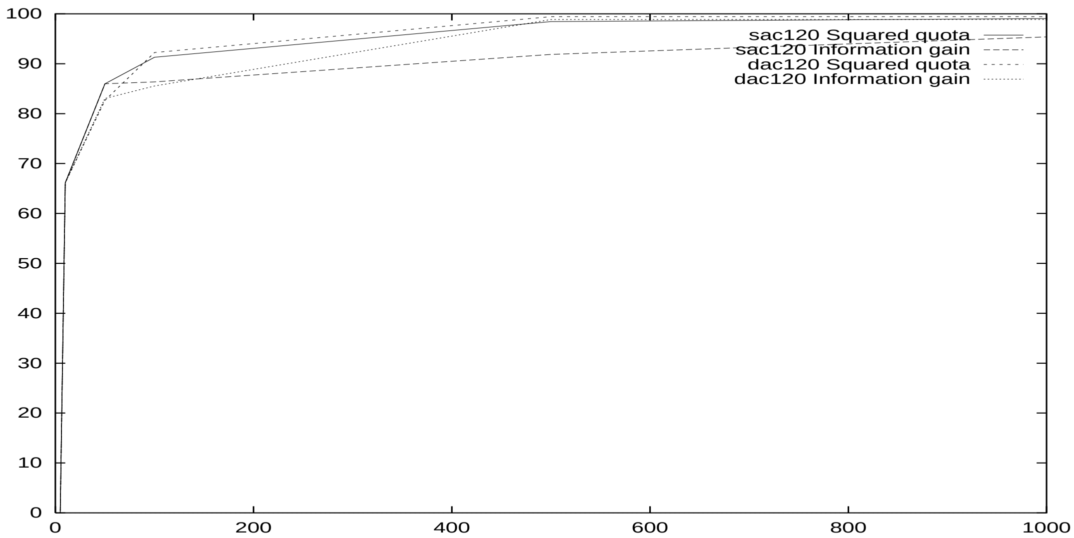

Figure 10: Accuracy in the King+Rook versus King+Pawn on a7 domain with windowsize 120.