Combining Knowledge Acquisition and Machine Learning to Control Dynamic Systems

G.M. Shiraz **C. Sammut**
School of Computer Science and Engineering
University of New South Wales
Sydney 2052
Australia

Abstract

This paper presents an interactive method for building a controller for dynamic systems by using a combination of knowledge acquisition and machine learning techniques. The aim is to build the controller by acquiring the knowledge of an operator skilled at that task. This method has been demonstrated for the skill of learning to fill an aircraft in a flight simulator. The simulator has been augmented to interact with a knowledge acquisition program for creating rules and logging the pilot's actions along with flight information. We have developed a method called *Dynamic Ripple Down Rules* for knowledge acquisition and *Learning Dynamic Ripple Down Rules* for automatically generating rules from the logged data. The rules were tested by running the flight simulator in autopilot mode where the autopilot code is implemented by the rules.

1 Introduction

In this paper, we report on an experiment that demonstrates how a combination of knowledge acquisition and machine learning techniques can be used to build a controller for a dynamic system. The aim is to build the controller by acquiring the knowledge of an operator skilled at that task. To demonstrate the effectiveness of this approach, a flight simulation program has been modified to interact with a knowledge acquisition program. A method, called *Dynamic Ripple Down Rules (DRDR)*, has been developed to acquire knowledge by interaction with the pilot and another system, *Learning Dynamic Ripple Down Rules (LDRDR),* has been used to generate rules automatically from data logged during a flight. The results have been tested by running the flight simulator in autopilot mode where the autopilot code has been replaced by the generated rules.

Due to complexity or lack of information about a plant, it is often difficult or impossible to construct a controller using classical methods. However, a competent human operator is often able to control a dynamic system. As a result, there is growing interest in mimicking the skills of the human operator (Michie et al, 1990; Sammut, et al 1992; Shiraz & Sammut 1995b; Urbančič & Bratko; 1994). Creating a set of rules that models the strategy used by the operator to control the task is one way to do this. The problem is how to determine a set of rules that captures the expert's strategy. Experiments have shown that it is often difficult for an expert to describe his or her strategy and the reason for choosing a particular strategy (Compton & Jansen, 1990). Sometimes it is even impossible for them to describe their strategies, especially when they are controlling a fast dynamic system such as an aircraft. The reason for this is that many skills are performed at a sub-cognitive level. These skills can be demonstrated, but are very hard to describe explicitly. Moreover, descriptions may be incomplete and approximate. Therefore, such descriptions can not be translated directly into an automatic controller. However, operator descriptions may contain general information which might be used as guidelines for constructing a controller automatically or semi-automatically.

In this paper, we propose a new interactive method, in which the expert (operator) and a learning program co-operate with each other to create the controller. The expert's description about his or her task is considered to be a set of general rules for controlling the system. These rules are refined by adding rules that are automatically created from the logged behaviour of the expert.

2 Previous Experiments

Michie et al (1990) in their pole-balancing experiments introduced the use of machine learning for learning to control dynamic systems. This method was later called *behavioural cloning*. In behavioural cloning, an operator skilled in a task is asked to control the system. During his or her performances, the state of the system, along with his or her actions, are logged into a file. This process is repeated several times until enough information is collected. In the next stage, a learning program uses the logged information to construct control rules for the system. Behavioural cloning was further refined by Sammut et al (1992) in their 'Learning to Fly' experiments and the technique has also been extended to other domains (Michie & Camacho, 1994; Shiraz & Sammut, 1995a; Urbančič & Bratko, 1994; Esmaili et al, 1995).

In addition to performing behavioural cloning experiments, in their 'Container Crane Controller' work, Urbančič & Bratko (1994), also recorded the advice that an operator

would give to a novice. They asked six volunteers to learn to control a crane simulator. After the operators had mastered the task, they asked the operator to write down their advice. They also encouraged them to discuss their experience and their written instructions with other operators to see if such discussions lead to improved performance.

One of the goals of this experiment was to see if there was any correspondence between the induced clone and the operator's instructions. It was found that the induced clone can uncover some of operator's subconscious control skill. They also reported how the verbal description of the operator can be used to improve the induced rules, when the induction program has failed to capture the behaviour for a particular condition. However, there was no systematic way of using the human operator's advice as background knowledge for machine learning technique. Moreover, often there was considerable divergence between the operator's description of the control strategy and the rules created by induction on that same operators performance.

Another approach to using an expert's advice during behavioural cloning is to combine a knowledge acquisition technique with induction (Shiraz et al, 1995c). While most of the skills involved in controlling a dynamic system are subcognitive and are therefore inaccessible to introspection, there are some aspects of those skills that may be explainable by an operator, particularly those involving high-level knowledge. By providing a suitable knowledge acquisition tool, this kind of knowledge can be documented in term of rules. As a result, the induction can be assisted to produce more transparent and robust clones.

They chose to modify Compton's (1989) Ripple-Down Rule (RDR) method for knowledge acquisition. RDR's provide a simple and powerful method for building a knowledge base interactively. The initial knowledge base usually consists of a simple rule for specifying what to do in the default case. When a rule is found to fail for a particular case, an exception rule is appended to the failed rule. The conditions required in the body of the exception rule can be easily determined by comparing the new case with cases that were previously handled successfully by the original rule.

In previous work (Shiraz & Sammut, 1995a) a controller that for flying a simulated aircraft through a predefined flight plan was constructed using a combination of RDR's and Gaines' (1992) Induct program. The use of an expert's advice resulted in the creation of rules that were more transparent and robust than those created by induction alone (Sammut et al, 1992). A major limitation of this work was that knowledge acquisition and machine learning were two separate tasks. In that experiment, following Sammut et al (1992), the flight was divided into seven different stages based on the predefined flight plan. For each stage, four separate RDR's for each of the four control actions (elevators, flaps, ailerons and throttle) were created. Each of these twenty eight RDR's was constructed either by using the RDR interactive knowledge acquisition method described above or by applying Induct to the logged data. In the work presented in this paper, we propose a method that merges knowledge acquisition and machine learning.

3 The Experiment

The flight simulator used in this experiment is one available on Silicon Graphics workstations. The pilot's task is restricted to flying a Cessna 150 through a predefined flight. The original program also gives the user the choice to fly different aircraft. The simulator was modified so that the pilot can run a trace of a flight and using the knowledge acquisition facility, diagnose and correct any incorrect decisions. These modifications will be described in more detail later.

The strategy a pilot uses for landing an aircraft is quite different from the strategy needed for straight and level flight. Sammut et al (1992) divided a flight into different stages according to the flight plan. The present experiments follow the same approach and use the same flight plan:

1. Take off : Take off and fly to 2000 feet.
2. Level out: Fly straight and level to a distance of 32000 feet from the airport.
3. Turn right: At a distance of 32000 feet[1], turn right to a compass heading of 330 degree.
4. Turn left: At a North/South distance of 42000 feet, turn left toward the runway.
5. Lining up: Change the aircraft's heading to line up on the runway.
6. Descending: descend to the runway.
7. Land: Land on the runway.

A modification of the original RDR knowledge acquisition method, Dynamic Ripple Down Rules (DRDR), is used for those parts of the flight where it was possible for the pilot to verbalise his or her strategy and, thus, write rules. However, some parts of the flight involve such subtle control strategies that it is very difficult, or even impossible, to verbalise those strategies. In those cases, LDRDR's (Learning Dynamic RDR's), an automatic rule generator, produces rules from the data logged data during a flight Figure 1 illustrates the architecture of the system.

3.1 Dynamic Ripple Down Rules (DRDR)

The basic form of a ripple-down rule is as follows:

if *condition* **then** *conclusion* **because** *case* **except**
  if *condition* **then** *conclusion* **because** *case* **except**
    if ...
else if ...

Initially an RDR may consist of the single rule:

if *true* **then** *default conclusion* **because** *default case*

---

1 Distances were chosen to correspond to features in the simulators landscape.

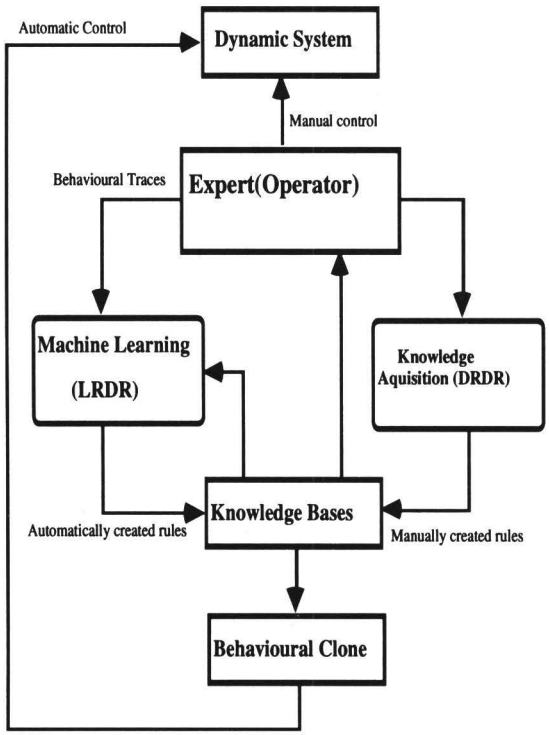

Figure 1: The basic algorithm for learning to control a dynamic system using a combination of knowledge acquisition (DRDR) and machine learning (LDRDR).

That is, in the absence of any other information, the RDR recommends taking some default action. For example, in a control application it may be to assume everything is normal and to make no changes. If a condition succeeds when it should not, then an exception is added (ie. a nested if-statement). Thus the initial condition is always satisfied so when the 'do nothing' action is inappropriate, an exception is added. If a condition fails when it should succeed, an alternative clause is added (i.e an else-statement). The new condition in the exception or alternative clauses is easy to determine.

With each condition/conclusion, RDR's store the *cornerstone case,* i.e. the case that caused the new condition to be created. When a new cases is incorrectly classified, it is compared with the cornerstone case of the incorrect condition and the differences are used to construct the new condition. Usually, the difference list is presented to the expert so that he or she may select the most relevant differences or generalise the conditions.

Originally, RDR's were developed for classification tasks such as medical diagnosis. To facilitate knowledge acquisition for controlling dynamic systems, additional facilities have been added:

Multiple knowledge bases: A pilot must often perform severl actions simultaneously, e.g., to turn the aircraft, the elevators and ailerons must be adjusted at the same time[2]. DRDR can manage several knowledge bases simultaneously. In this case, there is one for each of the four control actions.

New conclusions can be entered graphically: The user interface for a knowledge acquisition tool must be tailored for the application. For example, rules presented as text have little meaning for pilots. Therefore a graphical interface that provides analogues to an aircraft's instruments are important.

Transferring information from flight simulatot to DRDR: The pilot can pause a flight to investigate the RDR when it is not flying according to the pilot's wishes. When this happens information must be transferred from the flight simulator to the knowledge acquisition tool. Much of this information is then presented graphically.

3.2 Learning Dynamic Ripple Down Rules (LDRDR)

The manual knowledge acquisition of DRDRs is effective when it is possible for the expert to articulate rules about his or her performance. However, in many circumstances, it is very difficult for an expert to describe the control strategy. Often, attempts at such descriptions are incomplete and approximate and cannot be translated directly into an automatic controller. However, these descriptions may contain general information that can be used to guide the search that a machine learning system may conduct when trying to automatically build control rules from performance data.

3.2.1. Data preparation

The logged data usually contain information from different stages. These data are usually noisy and contain many redundant records. Pre-processing prepares the logs for the learning program. The pre-processing includes:

Segmenting the data: The data may contain information about a complete flight or only those parts of a flight where the pilot decided to record additional information. The first stage of pre-processing is to segment the data into the stages of the flight plan.

Discretising control values: The Cessna aircraft's control system consists of three control surfaces (elevators, flaps and ailerons) plus the throttle. The values of the control variables should be continuous. However, the learning algorithm can only deal with discrete class values, therefore the control variable values are discretised. In practice this is easy because the variables are only recorded to a limited precision and are, therefore, already discrete.

[2] To simplify the task, the rudder is ignored. Of course, this would not be appropriate for a real aricraft.

Eliminating spurious values: It is useful to eliminate intermediate values of the state variables so that the learning program is not swamped with spurious data. For example, when there is a change in elevator setting, eg. from 0° to 4°, all values in between are considered redundant and removed. Note that this preprocessing is vital, otherwise the number of rules generated by the learner will be extremely large.

Creating separate input for each control action: The data file from each stage contains information about all the manoeuvres performed by the pilot during that stage. Four separate files for each control action (elevator, flaps, rollers, throttle) are created for input to the learning program. To create a file for each control action, the attribute describing the control action is treated as the class variable and the rest of the attributes including attributes describing other control actions are treated as ordinary attributes.

3.2..2. The Learning Algorithm

The LDRDR algorithm constructs a controller as follows:

Inputs: current knowledge base; the behavioural traces; the priority list.

foreach record in the behavioural trace:

1. Test the next record against the knowledge base
2. If the conclusion of the RDR differs from the recorded trace, create a new rule to correctly handle the new record.

The condition part of a new rule is constructed by examining those variables which change most in correspondence with changes in control actions. The behavioural trace includes all the information that a pilot would normally see in the aircraft instruments. LDRDR tries to relate the pilot's actions to changes in the instrument readings and thereby predict what actions are required depending on the state of the instruments.

It is clear that a human expert only considers a relatively small number of variables at any time. To emulate this behaviour, LDRDR limits the number of conditions per rule. In order to avoid missing the important conditions in rule generation, LDRDR maintains a priority list of attributes for each control action. This list can be provided by the expert or created by the system. During learning this list is updated automatically by considering attributes that contribute more in rule generation.

After performing the pre-processing described previously, each of the data files and the existing RDR for a particular control action are used to extend the RDR using the LDRDR algorithm. LDRDR also uses the current priority lists.

The algorithm extends the RDR as follows:

foreach attribute in the priority list:

1. Compare the attribute's previous direction with its next direction. If there is a change in direction (e.g. it was increasing and becomes steady) then:
2. Create a test for the attribute. The test is based on the attribute's current value and its previous direction. The test always has the form:

   attribute **op** value

   where **op** is ">=" if the previous direction was increasing and "<=" if it was decreasing. *Value* is the value in the current record. The new test is applied to the cornerstone case associated with the last rule that was satisfied to make sure the test is correct for the current case but excludes the cornerstone case.
3. Add the test into a condition list.
4. Attributes in the priority list are ordered by a numerical score. Increment the attribute's priority.
5. If the number of tests in the condition reaches a user defined maximum, scan the rest of the attributes and just update their priorities if their direction has changed. The maximum is domain dependent and was set to 3 for these experiments.

end loop

If the condition list is not empty, create a rule and add it to the RDR. The conclusion of the rule comes from the action recorded in the trace. The current record becomes the rule's cornerstone case. The rule will be added as an exception to the last rule in the RDR if that rule is evaluated true. It is added as an alternative if false.

The output of LDRDR is an extension of the original RDR to cover cases in the input data that were not covered by the original RDR. The new RDR is converted into C if-statements by recursively traversing the RDR and creating one if-statement for each rule. An if-statement's conditions are the conjunctions of all true conditions in the RDR from the root to the rule.

4 Rule Construction

The learning task begins with some simple rules created by DRDR or by using LDRDR, applied to logged information. In both cases, the rules can be tested by running the simulation in autopilot mode where the autopilot code is derived from the rules. During the flight, if the aircraft does not behave as the pilot would expect, the pilot is able to pause the flight and trace the executed rules or the rules currently under execution. The pilot is also able to modify existing rules that seem incorrect by adding new rules. In this case, the previous and current situations of the aircraft, plus all the relevant flight variables, are presented to the pilot. As well, all the rules under execution for each control action are reported to the pilot. Whenever the flight is paused, there are several options available. Each of these options is described in the following sections.

4.1 Analysing rules

Information about the knowledge structure in the system is presented graphically to assist the pilot in understanding the current state of the knowledge base. The pilot may view the RDR either in text form as rules or graphically as a tree

structure drawn on the monitor. The tree may be traversed by moving the mouse over the tree.

4.2 Creating new rules

After evaluating the existing rules, if the pilot finds a conclusion is wrong or there is no interpretation for the current situation then he or she is able to add new rules. To help the pilot in creating new rules, the system provides a list of differences between the current state of the flight and the situation associated with the last condition that was satisfied. Choosing one or more variables from this list guarantees to produce a rule that will correctly interpret the new case but not the old one.

4.3 Flying the aircraft

This option is useful for the pilot to explore new flight plans or become familiar with existing stages. It can also be used by new pilots to get acquainted with the program. Moreover, during the learning task, pilots can use this option to fly some part of the flight manually and then put the aircraft in a specific state to investigate the behaviour of the autopilot at that situation or start recording data from that position.

4.4 Logging flight information

For those parts of the flight that are difficult for the pilot to diagnose a problem or suggest correct rules, he or she can simply show the correct action by switching to manual mode and flying the aircraft. During the flight, the pilot's actions, along with the flight information are logged to a data file. The flight's information is logged every second or when a change in a control action has been detected. These data can then be used to construct rules using LDRDR.

4.5 Flying on autopilot

In this mode, the flight simulator is controlled by an autopilot built by the system. The autopilot code is derived from the rules that have been created by the pilot during his or her previous flights and LDRDR from the logged data. This is used to test the rules.

5 Constructing an Autopilot

In this section we describe the results of a series of experiments with this system.

After every modification of the RDR, the flight simulation was run in autopilot mode to test the new RDR. To do this, the code of the original autopilot was replaced by the RDR (translated into C). A C function is also incorporated into the flight simulator to determine the current stage of the flight and when to change stages. The appropriate set of rules for each stage is then selected from four independent if-statements created in each stage for every control action (Sammut et al, 1992).

In the following section we describe a controller built for the first stage to demonstrate how these rules operate. Please note that these are the rules created by one pilot (either manually or by cloning). Another pilot would almost certainly create slightly different rules.

Stage 1

The rules for this stage are shown below, after translation to C:

ELEVATORS:

```
if (airspeed <= 50) return level_pitch;
if (elevation < 100) return pitch_up_3;
if (elevation < 110 && y_feet < 1970) return pitch_down_2;
if (elevation < 110) return pitch_up_1;
if (elevation < 130) return pitch_down_1;
if (elevation > 130) return pitch_down_3;
return level_pitch;
```

AILERONS:

```
if (azimuth <= 50) return left_roll_1;
if (azimuth <= 1800) return left_roll_2;
if (azimuth <= 3550) return right_roll_1;
if (azimuth <= 3599) return right_roll_1;
return no_roll;
```

FLAPS:

```
if (y_feet <= 250) return full_flaps;
if (y_feet <= 500) return half_flaps;
return no_flaps;
```

THROTTLE:

```
return throttle_100;
```

The rules for the elevator indicate that if the speed of the aircraft reached 50 knots, pull back on the stick to take off until the elevation increases to 10° (values in the rules are shown in tenths of degrees). Then push the stick forward to reach an elevation of 11 degrees. After that try to maintain the degree of elevation at 11° which is close to the value usually obtained by the pilot. When the aircraft reaches an altitude of 1970 feet decrease the elevator to get ready for the levelling out. Pitch-up-5 indicates a large elevator action and pitch-up-1 means a gentle elevator action. These elvator action names result from the discretisation of the actions, menioned previously.

The last statement is the default rule. It is the default action in that stage for the particular controller. For example the default rule for the elevator in all stages is level_pitch. Use of a default rule, results in a reduces number of rules.

The flaps rules states that as the aircraft reaches an altitude of 250 feet, decrease the flaps by half and if it reaches an altitude of 500 feet raise the flaps completely. The rules for the aileron keep the aircraft in the straight line. During take-off, the pilot who created these rules always applied full throttle.

6 Testing the system

The system tested using three volunteers: one of them was familiar with the system and another one was familiar with flight simulators. Their task was to create a set of knowledge bases using DRDR and LDRDR that can successfully complete the flight plan. described previously

Prior to the experiments, the subjects received a one hour tutorial in the use of the system. They also attended a demonstration. During the demonstration, the demonstrator explained how he flies the aircraft and the subjects were allowed to ask questions about the flight and the plan. The were allowed to practice until they become proficient in flying the aircraft.

During the experiments, the number of times that subjects interrupted the flight simulator, number of rules they created, the frequency use of DRDR and LDRDR and the amount of time they spent creating rules were recorded.

All the subjects were successful in creating a set of rules that could fly the aircraft through the given flight plan. The size of knowledge bases varied considerably from one subject to another one. One subject was cautious in creating rules for any possible state of the aircraft and thus produced a large RDR. Another subject was only interested in writing a set of rules that could fly the specific plan and so produced a much smaller RDR.

An interesting observation during these experiments was the reuse of knowledge. All the subjects tried to use rules created in an earlier stage if the task was similar. In particular, when creating rules for the elevator in stages three, four and five, they used the rules created for stage two and added some exception rules, if necessary.

Another observation was that the subjects began trying to use DRDR to manually construct rules, but when this became difficult, the subjects switched to LDRDR to build rules.

7 Conclusion

We have demonstrated that knowledge acquisition and machine learning techniques can be combined to successfully build control strategies for a dynamic system. We believe that this approach is applicable to other, similar domains, such as the container crane (Urbančič & Bratko, 1993) and others. In using "heuristics" such as limiting the number of conditions in each rule, we have followed the observation that experts rarely consider more than a few attributes when making a decision. Since we are trying to emulate the human trainer, it is reasonable to impose similar restrictions on the computer program.

All the testing up to this point has been in a noise-free simulation environment. The same technique should be able to handle noise, but further work is required to establish this claim. Further work is also required in improving the user interface. Ideally, pilots should never have to see any rules, but should be able to interact with the system entirely through the familiar surroundings of the cockpit. We also need more rigorous testing to measure the relative merits of LDRDR versus the more conventional induction programs previously used in behavioural cloning.

Acknowledgment

We thank Paul Compton and Philip Preston for their help in understanding and using RDR's in this project.

References

Compton, P. & Jansen, R. (1989). A Philosophical basis for knowledge acquisition. *3rd European Knowledge Acquisition for Knowledge Systems Workshop*, 1-17.

Compton, P. & Jansen, R. (1990). Knowledge in context: A Strategy for expert system maintenance. In C. Barter & M. Brooks (Eds.). *Proc AI 88, Lecture notes in Artificial Intelligence*, **406** (pp. 292-306). Berlin: Springer-Verlag.

Compton, P. (1992). Insight and Knowledge. In J. Boose, W. Clancey, B. Gaines, & A. Rappaport (Ed.), *AAAI Spring Symposium: Cognitive aspects of knowledge acquisition*, (pp. 57-63). Stanford University.

Esmaili, N., Sammut, C. & Shiraz, G. M., (1995). Behavioural Cloning in Control of a Dynamic System. In *The 1995 IEEE International Conference on System, Man and Cybernetics*, (pp. 2904-2909). Vancouver, Canada: IEEE.

Gaines, B. R., P. Compton (1992). Induction of Ripple-Down Rules. *Proceeding of the 5th Australian Joint Conference on Artificial Intelligence*, Hobart, Tasmania.

Michie, D., Bain, M., and Hayes-Michie, J. E. (1990). Cognitive model from subcognitive kills. In M. Grimble, S. McGhee, and P. Mowforth (Eds.) *Knowledge-base Systems in Industrial Control*, Peter Peregrinus.

Michie, D. & R. Camacho (1994). Building Symbolic Representations of Intuitive Real-Time Skills from Performance Data. In *Machine Intelligence 13*, Eds. K. Furukawa, D. Michie, and S. Muggleton, pp. 1-30.

Sammut, C., S. Hurst, D. Kedzier. , D. Michie (1992). Learning to Fly. In *Proceeding of the 9th International Workshop on Machine Learning*, edited by D. Sleeman and P. Edwards, Morgan Kaufmann, 385-393.

Shiraz, G. M. & Sammut, C. (1995a.) An Incremental Method for Learning to Control Dynamic Systems. In *The Machine Learning Workshop of the IJCAI-95*, (pp. 139-175). Montreal, Canada: IJCAI.

Shiraz, G. M. & Sammut, C. (1995b). Learning to Fly in the Presence of an Expert. In *The third ICEE-95 (Control)*, 3 (pp. 77-85). Tehran, Iran: Iran University of science and Technology Press.

Shiraz, G. M., Sammut, C. & Esmaili, N. (1995c). Man, Machine Cooperation for Learning to Control Dynamic Systems. In *The 1995 IEEE International Conference on System, Man and Cybernetics*, (pp. 1108-1112). Vancouver, Canada: IEEE.

Urbančič, T. & Bratko, I. (1993). Learning to control dynamic systems. *Machine Learning and Statistical Classification*, D. Michie(ed.), Ellis-Horwood.

Urbančič, T. & Bratko, I. (1994). Reconstructing human Skill with Machine Learning. *11th European Conference on Artificial Intelligence*, Ed. A. Cohn, 489-502.