# Advanced LCS

GAassist

Dr. J. Bacardit, Dr. N. Krasnogor

# Objectives of GAssist

- Generation of compact and accurate solutions
  - Allows the system to learn better
  - Generate compact and interpretable solutions
  - Avoid over-learning
  - Achieved by:
    - Explicit default rule mechanisms
    - Initialization policies
    - MDL-based fitness function

# Objectives of GAssist

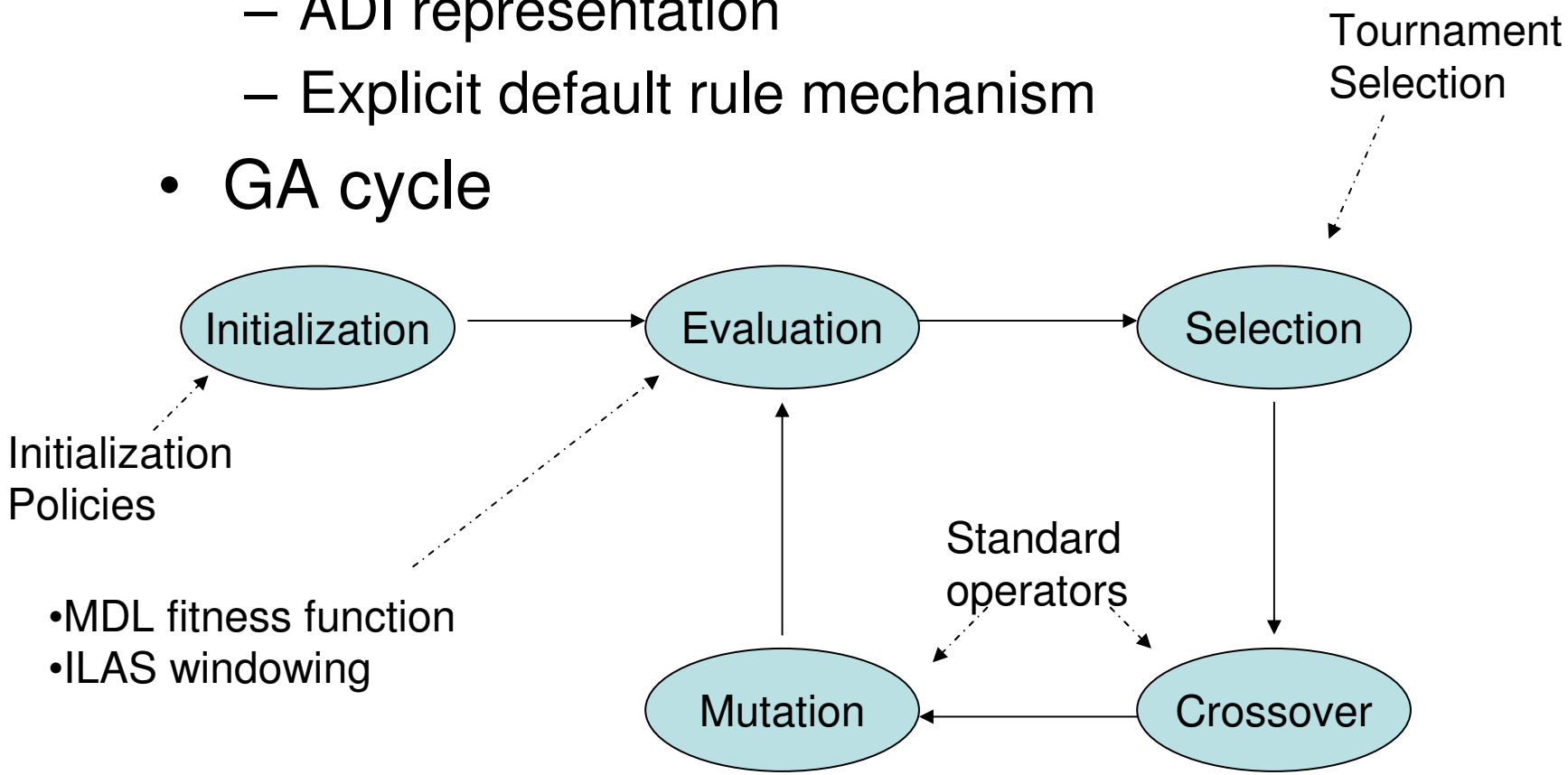
- Run-time reduction
  - Genetic Algorithms and LCS in general are quite slow
  - Moreover, our aim is to apply GAssist to hugh datasets
  - We need some method to alleviate the computational cost of the system
  - Achieved by a Windowing technique called ILAS

# Framework and workflow

- Individuals are interpreted as a decision list: an ordered rule set
- The semantically correct crossover operator of GABIL is used
- The GABIL representation is used for nominal attributes
- C++ and Java versions of the system. Java version available at http://www.asap.cs.nott.ac.uk/~jqb/PSP/GAssist-Java.tar.gz

# Framework and workflow

- Representation
  - ADI representation
  - Explicit default rule mechanism
- GA cycle



# The ADI rule representation

- Handling real-valued attributes by using discretization
  - **Discretization:** converting a continuous variable into a discrete variable with finite number of elements
  - There is no discretization method suitable for all datasets, because each algorithm introduces bias
  - Are all cut-points relevant?
  - ADI representation handles these two issues

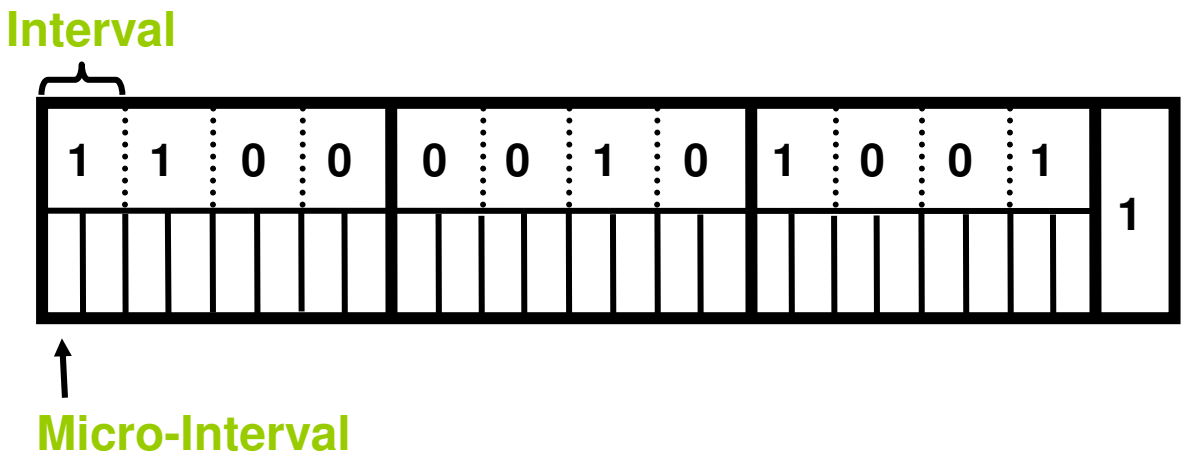
# The ADI rule representation

- ADI knowledge representation is based on GABIL
  - Predicate $\rightarrow$ Class
  - Predicate: Conjunctive Normal Form (CNF) $(A_1=V_1^1 \vee .. \vee A_1=V_1^n) \wedge ..... \wedge (A_n=V_n^2 \vee .. \vee A_n=V_n^m)$
    - $A_i$ : ***i**th* ***a***ttribute
    - $V_i^j$ : ***j**th* ***v***alue of the **i***th* attribute
  - The rules can be mapped into a binary string

**1100|0010|1001|1**

# The ADI rule representation

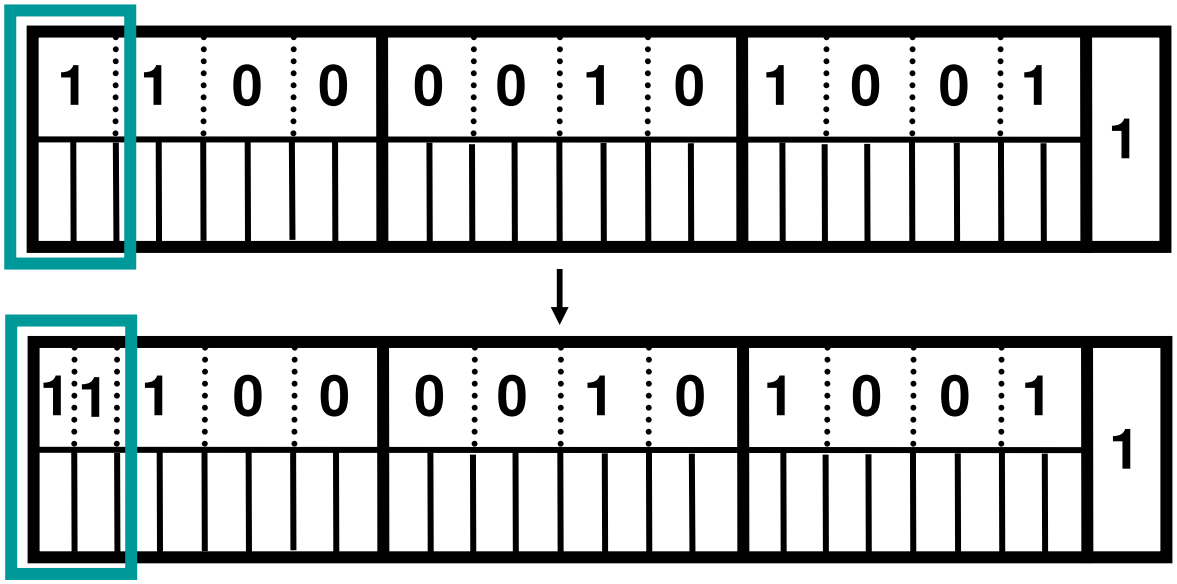
- The ADI representation generates the semantic values of the attributes (intervals) through the evolutionary process of the GA
- These intervals are build over a set of base intervals generated by a discretization algorithm (called **micro-intervals**).
- This discretization can be of any kind



# The ADI rule representation

- How these intervals are evolved?
  - The intervals can split or merge

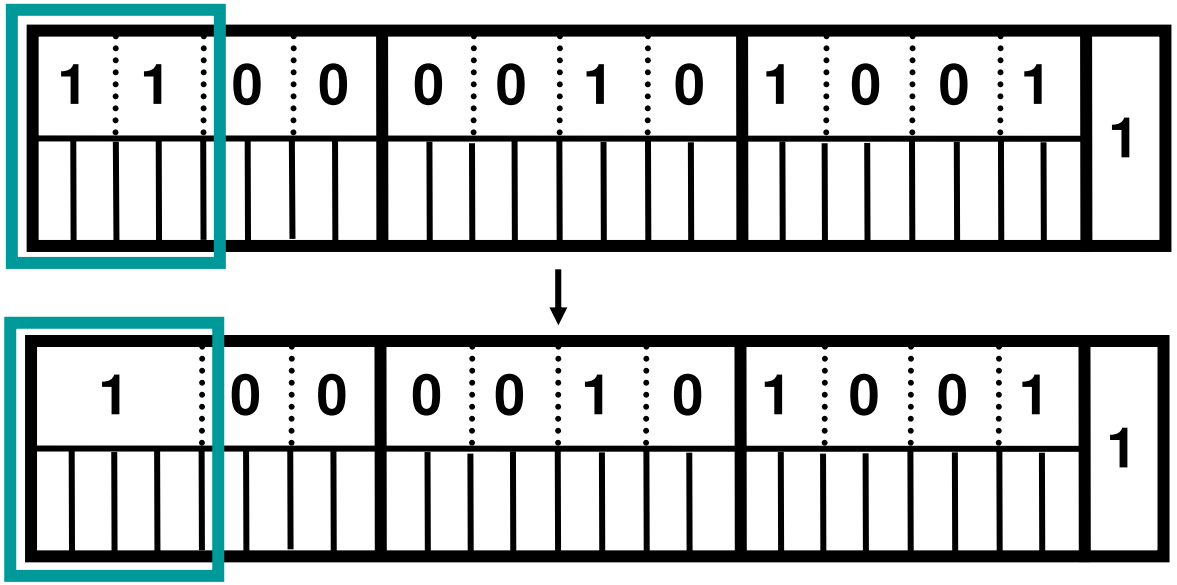
**Split**



# The ADI rule representation

- How these intervals are evolved?

**Merge**



# The ADI rule representation

- We have a predefined pool of discretization algorithms
- Initialization assigns randomly a discretizer to each attribute-term of each rule of each individual
- In this way, the system is not completely tied to the bias introduced by each single discretization algorithm

# Generation of compact and accurate rule sets

- The default rule mechanism
  - When we encode this rule set as a decision list we can observe an interesting behavior: the emergent generation of a default rule
  - Using a default rule can help generating a more compact rule set
    - Easier to learn (smaller search space)
    - Potentially less sensitive to overlearning
  - To maximize this benefits, the knowledge representation is extended with an explicit default rule

# Generation of compact and accurate rule sets

- What class is assigned to the default rule?
  - Simple policies such as using the majority/minority class are not robust enough
  - We can combine the simple policies, but doubling the run-time
  - Automatic determination of default class

# Generation of compact and accurate rule sets

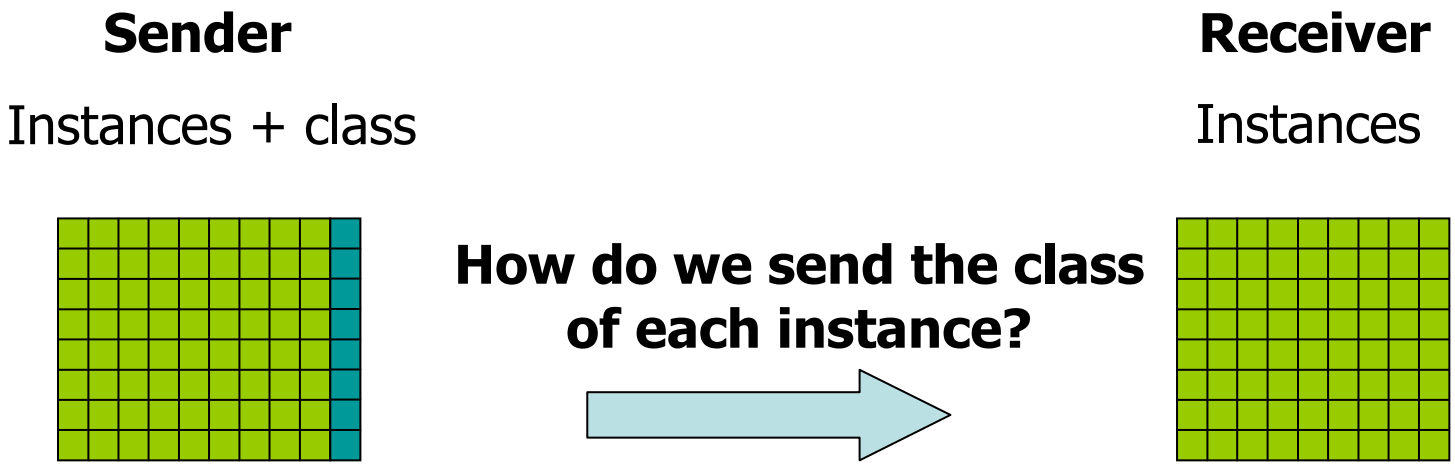
- Automatic default class
  - All default classes compete in the population
  - A niching mechanism is used to guarantee a fair competition: different default classes might have different learning rate
  - The niching mechanism is disabled when all niches are equally good (in accuracy)

# Generation of compact and accurate rule sets

- Initialization policy: covering operator
  - Inspired in the covering operator of XCS
  - Each rule initialization samples an instance from the training set
  - Two methods of sampling instances from the training set
    - Uniform probability for each instance
    - Class-wise sampling probability

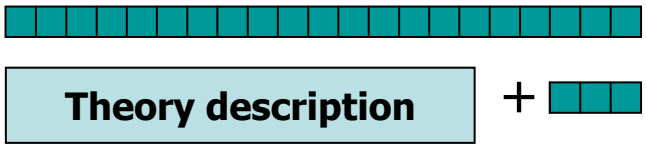
# Generation of compact and accurate rule sets

- MDL-based fitness function




1) Sending the classes

2) Generating a theory and sending it plus its exceptions



# Generation of compact and accurate rule sets

- The solution that minimizes the length of sending the theory + its exceptions will be the best one
- MDL based fitness function is the minimization of the following formula:

$$MDL = W \cdot TL + EL$$

- W is domain specific. In order to avoid a manual tuning we have also developed an adaptive heuristic process to tune this parameter

# Generation of compact and accurate rule sets

- TL depends on the knowledge representation used. We should not select the shortest theory length definition, but a definition that promotes well-generalized solutions
- This is the most interesting feature of this fitness function. We can explore more efficiently the search space by guiding the search based on the content of the individuals
- TL metric will promote
  - Individuals with few rules
  - Individuals with few expressed attributes

# Run-time reduction

- A windowing mechanism named Incremental Learning with Alternating Strata (ILAS) is used
- The mechanism uses a different subset of training examples in each GA iteration