Toward an Evolutionary Computing Modeling Language

Heiko Aydt, *Student Member, IEEE,* Stephen J. Turner, *Member, IEEE,* Wentong Cai, *Member, IEEE,* Malcolm Yoke Hean Low, *Member, IEEE,* Yew-Soon Ong, *Member, IEEE,* and Rassul Ayani, *Member, IEEE*

***Abstract*—The importance of domain knowledge in the design of effective evolutionary algorithms (EAs) is widely acknowledged in the meta-heuristics community. In the last few decades, a plethora of EAs has been manually designed by domain experts for solving domain-specific problems. Specialization has been achieved mainly by embedding available domain knowledge into the algorithms. Although programming libraries have been made available to construct EAs, a unifying framework for designing specialized EAs across different problem domains and branches of evolutionary computing does not exist yet. In this paper, we address this issue by introducing an evolutionary computing modeling language (ECML) which is based on the unified modeling language (UML). ECML incorporates basic UML elements and introduces new extensions that are specially needed for the evolutionary computation domain. Subsequently, the concept of meta evolutionary algorithms (MEAs) is introduced as a family of EAs that is capable of interpreting ECML. MEAs are solvers that are not restricted to a particular problem domain or branch of evolutionary computing through the use of ECML. By separating problem-specific domain knowledge from the EA implementation, we show that a unified framework for evolutionary computation can be attained. We demonstrate our approach by applying it to a number of examples.**

***Index Terms*—Evolutionary algorithms, high-level languages, high-level modeling, UML.**

I. Introduction

SPECIALIZATION is an important issue for solving optimization problems. While general-purpose algorithms can be applied to a larger variety of problems, they are generally outperformed by algorithms that have been specifically designed to solve a particular problem. However, an algorithm that outperforms another algorithm for solving a particular class of problems will perform worse for solving at least one other class of problems. This insight is known as the *no free lunch* theorems [1]. Essentially these theorems state that the average performance of an algorithm over all classes of problems remains constant. A performance advantage for a particular class of problems can only be achieved by accepting a degraded performance for at least one other class of problems. Although it has been shown that the no free lunch theorems do not necessarily apply under certain circumstances [2], [3], their implications on the design of problem solving algorithms are well recognized.

The importance of using domain knowledge in the design of effective evolutionary algorithms (EAs) is widely acknowledged, not only as a consequence of the no free lunch theorems, but also as a result of experience with practical applications in this area. Specialization is achieved mainly by embedding (i.e., hard-coding) available knowledge about a problem into the algorithms. An overview on the various ways to incorporate domain knowledge into EAs was described by Bonissone *et al.* [4]. Domain experts use their knowledge about a problem to choose a representation that ideally captures all the important features of the problem. In addition, they design specialized operators and choose parameters that are known to suit the problem at hand. This has led to a large variety of (sometimes highly) specialized EAs and meta-heuristics that have been developed to solve a particular problem.

Adapting a specialized EA to another problem may be difficult and thus not desirable in many cases. For example, different problems may require different operators and/or genotype representations. In the last few decades, a plethora of EAs has been manually designed by domain experts for solving specific problems. Many noteworthy EA libraries have been made freely available and used to construct problem-specific EAs. For example, this includes the GAlib,[1] EOlib[2] [5], and EASEA[3] [6]. The presence of EA libraries helps to reduce the amount of programming efforts and time required by providing commonly used genotype representations and evolutionary operators for rapid development of EAs.

Although EA libraries can be used to construct different algorithms, they do not provide the means to describe EAs on an abstract level, i.e., to create blue-prints that highlight the characteristic features of EAs. Domain knowledge used to design an EA is reflected by its implementation (i.e., source code) and thus inseparably embedded into the EA. A

Manuscript received June 18, 2009; revised November 30, 2009, April 14, 2010, and August 30, 2009; accepted September 12, 2010. Date of publication January 10, 2011; date of current version March 30, 2011.

H. Aydt, S. J. Turner, W. Cai, M. Y. H. Low, and Y.-S. Ong are with the School of Computer Engineering, Nanyang Technological University, Singapore (e-mail: aydt0001@e.ntu.edu.sg; steve@pmail.ntu.edu.sg; aswtcai@ntu.edu.sg; yhlow@ntu.edu.sg; asysong@ntu.edu.sg).

R. Ayani is with the School of Information and Communication Technology, Royal Institute of Technology, Stockholm SE-100 44, Sweden (e-mail: ayani@kth.se).

Digital Object Identifier 10.1109/TEVC.2010.2081368

[1] Available at http://lancet.mit.edu/ga.

[2] Available at http://eodev.sourceforege.net.

[3] Available at http://sourceforge.net/projects/easea.

different approach would be to separate domain knowledge from the implementation of the EA. Given: 1) a high-level description language to specify relevant problem-specific meta information, and 2) an appropriate algorithm that is capable of interpreting the language, EAs can be constructed that are not *a priori* specialized for a particular problem but capable of exhibiting specialized behavior based on the meta information provided about the problem at hand.

An example for high-level approaches is grammar-based approaches, such as grammar-based genetic programming [7] or grammatical evolution [8]. Different problems can be supported by providing problem-specific grammars. For example, Jung and Reggia [9] presented a descriptive encoding language that makes use of a problem-specific grammar which can be used to evolve designs of neural networks. Their high-level descriptive language, however, is specially designed for neural networks and is thus not directly applicable to other domains. A more general approach is described by Veenhuis *et al.* which allows specification of EAs on an abstract level using XML [10], [11]. Although this includes specification of genotypes and various algorithmic aspects (e.g., operators), it has several shortcomings. For example, it does not support grammars and constraints.

Existing approaches either do not consider any form of high-level specification or have been designed for a particular problem or branch of evolutionary computing. Although existing approaches, most notably grammar-based approaches, address some of these issues, a unified approach remains an open issue. For example, grammar-based approaches have been mostly used for genetic programming and it is not clear how applicable they are to problems from other domains. A formal general-purpose language that can be used across all branches of evolutionary computing to model problems across different domains would thus be a first step toward a unifying framework.

We present a general-purpose approach that can be used across the different branches of evolutionary computing and address some of the limitations of existing approaches. For this purpose, we propose a formal evolutionary computing modeling language (ECML), based on the unified modeling language (UML) [12], that allows the design of problem-specific genotypes. This includes specification of the structure and encoding of genotypes as well as relevant details regarding the processing of genotypes. In addition to the proposed ECML, we introduce the concept of meta evolutionary algorithms (MEAs) as a class of EAs that is capable of interpreting ECML-based genotype models and processing genotypes accordingly. MEAs are meta algorithms because their search mechanisms and behavior are defined by the meta information specified in the genotype model. Furthermore, MEAs can be designed as interpreters which allow them to dynamically adapt their behavior as the meta information is changing.

The remainder of this paper is structured as follows. In Section II, we give an overview of different kinds of meta information that is considered by ECML. In Section III, we introduce ECML which can be used to formally specify problem-specific genotype models. In Section IV, we describe the concept of MEAs and introduce our proof-of-concept implementation MEAα. Subsequently, we apply ECML and MEAα to several complex optimization problems, including a dynamic optimization problem to demonstrate the flexibility and efficacy of the proposed approach in Section V. This is followed by a discussion and our conclusions in Sections VI and VII, respectively.

II. Meta Information: Genotype Representations, Operators, and Constraints

An important issue for improving the performance of an EA is the selection of an appropriate genotype representation [13]. A good representation captures important features of the problem domain that are necessary to effectively solve the problem [39]. To date, there exist a large variety of genotype representations. For example, string and matrix structures are commonly found in the context of genetic algorithms, evolutionary programming, evolutionary strategies, and learning classifier systems. In genetic programming, on the other hand, tree structured genotypes are used. Encoding of genotypes includes binary-, integer-, real-, and mixed coding. More recently, some work has encoded information regarding the logical structure as part of the genotype. For example, analog genetic encoding [14], used to evolve analog circuits and networks (hence the name), incorporates special tags into the genotype to indicate the type of information that follows. Nevertheless, meta information regarding the structure and encoding of the genotype is usually not provided.

Most components of an EA require knowledge about the genotype and its representation. For example, this includes variation operators, i.e., recombination and mutation operators. In addition to selecting an appropriate representation it is also important to select appropriate operators that suit the problem and the chosen genotype representation. Some variation operators are more generally applicable than others. For example, a uniform crossover operator can be applied regardless of how genotypes are encoded. In contrast, arithmetic crossover operators are only useful for integer-coding or real-coding genotypes. There are also EA components that generally do not require any knowledge about the genotype. For example, selection operators are concerned with the quality of individuals (in terms of some fitness metric) and thus do not require knowledge about the genotype representation.

Another important issue that has to be considered is constraint handling. There are various methods that are commonly used for constraint handling. For example, this includes penalizing or repairing genotypes that do not satisfy constraints. Another constraint handling method is to use an indirect encoding of solutions to the problem. Moreover, it is also possible to make use of knowledge about constraints to design operators that prevent the generation of genotypes that do not satisfy the constraints altogether. This method requires the EA to be aware of constraints. For example, in the context of high-level approaches, attribute grammars can be used to specify semantic information that is exploited by the EA when evolving the genotypes [15].

Domain knowledge that has been used at design time to construct an EA is reflected by the choice of genotype

representation, operators, and constraint handling methods. As explained above, we aim at separating this kind of knowledge from the actual implementation. We believe that the genotype representation is essential in this regard as many components of an EA rely on knowledge about the genotype. For this reason, we have chosen a genotype-centric approach and focus on the specification of genotype models using ECML. This not only includes meta information regarding the structure and encoding of genotypes but also meta information regarding the various processing aspects and constraints.

III. Genotype Specification

We aim at proposing a formal modeling language which can be widely accepted within the community. UML is a general-purpose modeling language and has been chosen as the basis for ECML because it is a formal language (all elements have a well-defined meaning), it is an industry standard, extensible, and supports constraints. In addition, UML is neutral because it is not associated with a specific problem or branch of evolutionary computing. An introduction to UML, including references to the corresponding literature, can be found in [16].

UML provides a formalism that allows the modeling of the various artifacts of a system. This formalism includes various standardized modeling elements and a well-defined notation. Although its original purpose is to model software systems, UML can also be used to model non-software systems. For example, the systems modeling language (SysML) is a general-purpose language that uses a subset of UML elements to support the specification, analysis, design, and verification of complex systems [17]. Similarly, ECML is a general-purpose language that is used to model genotypes and related processing aspects of an EA. For this purpose, ECML uses basic UML elements and introduces extensions that are specially needed in the context of evolutionary computing.

Two important advantages of UML are the support of constraints and its extensibility by means of stereotypes and profiles.

1) Constraints may be expressed as plain text or by formal means. For example, the object constraint language [18], which supplements UML, can be used for this purpose. Constraints are enclosed by braces (i.e., {constraint}) and attached to the corresponding modeling element.
2) Stereotypes are used to extend the core semantics of UML with semantics that are needed in the context of a particular domain. A stereotype is enclosed by guillemets (i.e., «stereotype») and is attached to the corresponding modeling element. By doing so, the modeling element is marked as the one that has a specific meaning in the domain context.
3) Profiles are specifications of common model elements within a specific domain context. For example, SysML is defined as a profile which extends UML by a set of elements that are specifically needed for modeling complex systems.

In the remainder of this section, we will describe the various modeling elements that are specially needed in the context of evolutionary computing. We also introduce a number of semantic extensions using corresponding stereotypes.

A. Genotype Encoding and Structure

A genotype model defines the structure and encoding of genotypes in the context of a particular problem. ECML provides two kinds of modeling elements that can be used to specify genotype models: entities and relationships.

1) *Entities:* Entities are used to define the various building blocks, such as genes, that are needed to model genotypes for a particular problem.

A gene is defined as "a locatable region of genomic sequence, corresponding to a unit of inheritance, which is associated with regulatory regions, transcribed regions, and/or other functional sequence regions" [19]. Alleles are the different possible forms of a gene. Phenotypic traits depend on how corresponding genes are expressed, i.e., which alleles are present at the particular locations of the genome that are associated with the corresponding genes. For example, the eye color in humans is a phenotypic trait and depends on how the genes, that are associated with encoding the eye color, are expressed by corresponding alleles (e.g., blue or green). A genome is the complete genetic information of an individual.

Similarly, in the context of evolutionary computing, we consider genes as the basic building blocks that constitute the genotype of an individual. In addition, alleles are used for coding purposes. A genotype consists of a number of genes that, depending on their location in the genotype, are associated with different problem-specific features. They are identified by the «gene» stereotype. The notation for genes is illustrated in Fig. 1(a). For example, consider the family of knapsack problems [20]. This kind of problem is concerned with the optimal selection of items, each being associated with a profit and a weight, in order to maximize the overall profit while not exceeding a total weight limit. Genotypes for the knapsack problem can be modeled by using genes to encode the selection of items. Each gene is expressed by an allele that indicates whether a particular item is selected or not. The genotype is thus a sequence of "yes" and "no" alleles in which the ith allele corresponds to the ith item.

We distinguish between two types of alleles: symbolic alleles and numerical alleles. For example, in the knapsack problem described above, symbolic alleles are used to indicate selection. Unlike symbolic alleles, numerical alleles can be used to perform arithmetic operations. Valid numerical values for expressing a particular gene are modeled by an interval with lower bound lb, upper bound ub, and resolution r. The resolution is used to specify the accuracy of the numerical values. An interval thus represents a set of numerical values $\{lb, lb+r, lb+2r, \ldots, ub-2r, ub-r, ub\}$. We further distinguish between integer-value and real-value numerical alleles, identified by the «int(lb,ub,r)» and the «real(lb,ub,r)» stereotypes, respectively. Symbolic alleles are considered as default type and thus not explicitly marked as such. The notations for the various types of alleles are illustrated in Fig. 1(b)–(d).

Multiple building blocks can be grouped together and treated as a higher-order building block. For this purpose,

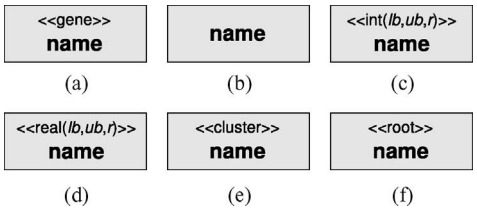


Fig. 1. Notation of modeling elements used for (a) genes, (b) symbolic alleles, (c) integer-value alleles, (d) real-value alleles, (e) cluster entities, and (f) root entities.

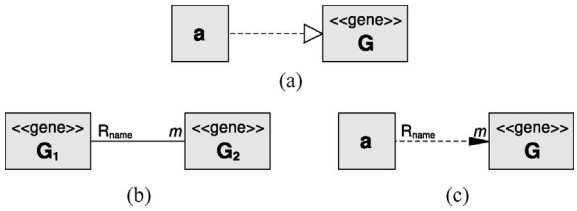


Fig. 2. Notation of relationships used in the genotype model for (a) expression, (b) association, and (c) activation.

the concept of clusters is introduced. Unlike genes, cluster entities are not used for coding purposes, i.e., they are not expressed by alleles. Instead, they are used to represent higher-order building blocks. Cluster entities play an important role for structuring genotypes. For example, later in Section V-A we discuss a representation for a job-shop scheduling problem that makes use of cluster entities to structure the genotype. A cluster entity is identified by the «cluster» stereotype. A cluster with a special meaning is the root entity which indicates the starting point for interpreting the genotype model. It has to appear exactly once in a genotype model and is identified by the «root» stereotype. The notations for cluster and root entities are illustrated in Fig. 1(e) and (f), respectively.

2) *Relationships:* ECML defines three kinds of relationships: expression, association, and activation relationships.

a) *Expression relationships:* When specifying the genotype model for a particular problem, it is necessary to indicate which alleles can be used to express a certain gene. For this purpose, the expression relationship is used. It connects an allele with a gene. Depending on the problem, there might be several alternative alleles for expressing a gene. For example, in the knapsack example above, a gene can be expressed by either one of two possible alleles. In such a case, there can be multiple expression relationships, involving the same gene and different alleles, to define the possible alternatives for expressing the gene. The notation for expression relationships is illustrated in Fig. 2(a). This example illustrates a gene G which can be expressed by a symbolic allele a.

b) *Association and activation relationships:* Many problems require the definition of multiple genes of the same kind. For example, consider the knapsack problem described above. Depending on the number of selectable items, there has to be a corresponding number of genes in the genotype. For this purpose, association relationships are used. They establish one-to-many relationships between two building block entities (i.e., gene or cluster entity). This allows us to specify the multiplicity of a building block relative to another. The exact multiplicity is an attribute of the relationship. The notation for association relationships is illustrated in Fig. 2(b). This example illustrates two genes G_1 and G_2 which are associated with each other. For each instance of gene G_1 there are exactly m instances of gene G_2. The naming convention for association relationships is R_{name}, where *name* is the name of the relationship.

Another relationship, which is similar to an association relationship, is the activation relationship. It indicates whether a particular building block is present in the genotype or not, depending on the expression of a particular gene. Unlike an association relationship, which connects two gene or cluster entities with each other, an activation relationship connects an allele with a gene or cluster entity. Similarly to association relationships, it is possible to specify the multiplicity of an activation relationship. The notation for activation relationships is illustrated in Fig. 2(c). This example illustrates an allele a which activates m instances of gene G. For each appearance of allele a there are m instances of gene G in the genotype. The naming convention for activation relationships is the same as for association relationships.

3) *Example Genotype Models M_1 and M_2:* The genotype model M_1, illustrated in Fig. 3(a), consists of two genes G_1 and G_2. Corresponding expression relationships indicate that gene G_1 can be expressed by the symbolic alleles a or b. Furthermore, it is indicated that gene G_2 can be expressed by the symbolic alleles c or d. There are three instances of gene G_1 in every genotype instance created, based on this model. This is indicated by the association relationship R_1 which connects the root entity M_1 with gene G_1. Whether or not there will be any instances of gene G_2 in the genotype instance depends on how the various instances of gene G_1 are expressed. For each instance of gene G_1 which is expressed by symbolic allele b, there are two more instances of gene G_2. This is indicated by the activation relationship R_2.

Although the size of genotype instances for genotype model M_1 is not fixed, it is limited to a maximum of three G_1 and six G_2 instances. In contrast, the genotype model M_2, illustrated in Fig. 3(b), can grow arbitrarily large during the process of evolution. Genotype instances based on genotype model M_2 have at least three G_1 instances (because of association relationship R_1). For each G_1 instance there is a G_2 instance (because of association relationship R_2) and depending on how this gene is expressed (either y or n), there are more G_1 instances (because of activation relationship R_3). Circular relationships, such as in genotype model M_2, effectively allow the genotype structure to grow during the process of evolution. This makes it possible to apply our approach in genetic programming and grammatical evolution (see Section V-C for details).

The actual genotype representation depends on the EA implementation. Later in Section IV-A we discuss our proof-of-concept MEA implementation and explain how these two examples are represented.

B. Genotype Processing Meta Information

The various entities and relationships described above can be used to model the problem-specific genotype structure

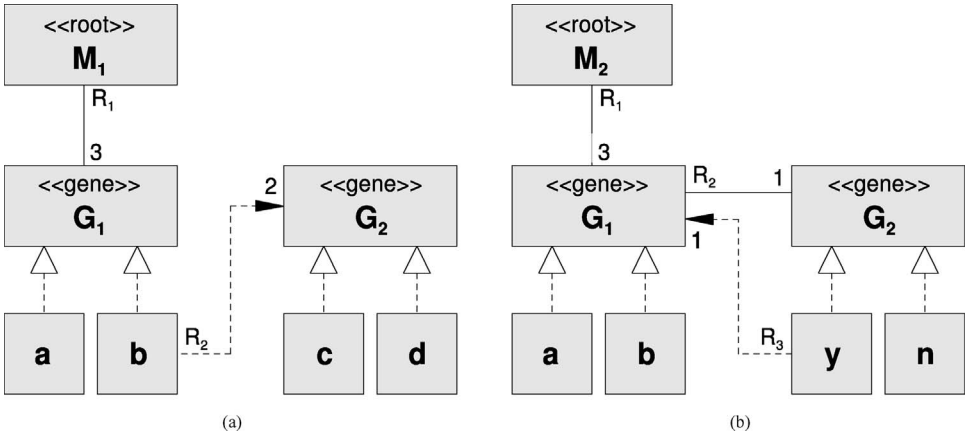


Fig. 3. Example for genotype models using (a) non-circular relationships and (b) circular relationships.

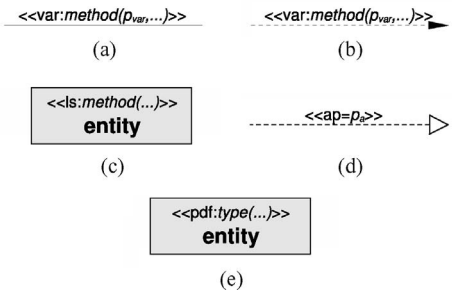


Fig. 4. Notation of extensions for specification of meta information regarding processing aspects: "var" stereotypes assigned to an (a) association relationship and an (b) activation relationship, a (c) "ls" stereotype assigned to an entity, an (d) "ap" stereotype assigned to an expression relationship, and a (e) "pdf" stereotype assigned to an entity.

and its encoding. In addition, a number of extensions are introduced in order to specify meta information regarding the various processing aspects of an EA.

1) *Variation Operators:* An important aspect of an EA is the creation of new individuals. This is done by applying variation operators (i.e., recombination and mutation) to selected individuals of the population. What kind of variation operations to use depends on the problem. ECML allows to assign variation operators to association and activation relationships by using a corresponding stereotype. The "var" stereotype is defined as «$\texttt{var}_{type}\texttt{:}\,method\,(\,p_{var}, v_0, \ldots, v_{n-1}\,)$», where *method* indicates the particular kind of variation operator, p_{var} is the probability that the method is performed, and $v_0, \ldots, v_{n-1}$ are optional parameters that might be required by the method. With *type* we further distinguish between "var" stereotypes for recombination «$\texttt{var}_{\texttt{rc}}\texttt{:...}$» and mutation «$\texttt{var}_{\texttt{mt}}\texttt{:...}$».

Multiple variation operators might be applicable to the same relationship. For this purpose, it is possible to attach multiple "var" stereotypes to the same relationship. In addition, it is possible to specify different recombination operators for different association or activation relationship. Effectively this means that different variation operations are performed for different parts of the genotype. This applies to recombination operators and mutation operators likewise. If no variation operator is specified along a particular relationship, then there will be no such operation performed for the corresponding part of the genotype. The notation of the "var" stereotype, attached to an association relationship and an activation relationship, is illustrated in Fig. 4(a) and (b), respectively.

2) *Local Search:* EAs are sometimes used in combination with a local search method in order to improve the performance. These so-called memetic algorithms [21] have become popular because their performance is often significantly better as compared to EAs that do not perform local search. Depending on the problem, different kinds of local search methods have to be used. For this purpose, we introduce the "ls" stereotype which indicates that a specific local search method is applied. This stereotype is defined as «$\texttt{ls}\texttt{:}\,method(p_{ls}, v_0, \ldots, v_{n-1})$», where *method* indicates the particular kind of local search method, p_{ls} is the probability that the method is performed, and $v_0, \ldots, v_{n-1}$ are optional parameters that might be required by the method. The notation of the "ls" stereotype is illustrated in Fig. 4(c).

3) *Allele Probability Distribution:* When creating or mutating genotypes, alleles are often chosen randomly according to some probability distribution. Usually, these probabilities are uniformly distributed among all possible alleles. It is possible to specify a non-uniform allele probability distribution by using domain knowledge. For this purpose, we introduce the "ap" stereotype which is attached to any expression relationship and allows the specification of a non-uniform allele probability distribution. It is defined as «$\texttt{ap}\texttt{=}p_a$», where p_a is the probability of the specific allele to be selected. As a result, some alleles are selected more frequently than others. This effectively causes the search to be biased toward a certain region in the search space. The notation of the "ap" stereotype is illustrated in Fig. 4(d).

In case there are multiple numerical alleles for a gene, then one allele is selected with a corresponding probability p_a. Effectively, this means that a particular interval is selected with a probability p_a. However, for the selected interval, all possible values are equally likely. This means that a uniform probability distribution function is implicitly assumed for selecting a specific value within a particular value range. Alternatively, it is possible to specify a customized

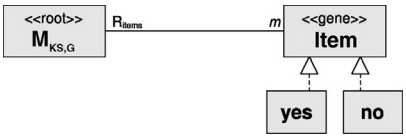


Fig. 5. General genotype model $M_{KS,G}$ of the knapsack problem, defining the structure and encoding of the genotype.

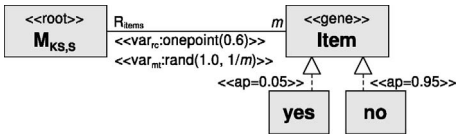


Fig. 6. Specialized genotype model $M_{KS,S}$ of the knapsack problem featuring stereotypes that indicate the choice of variation operators and a customized allele probability distribution.

probability distribution function using the "pdf" stereotype. It is defined as «pdf:*type*($v_0, \ldots, v_{n-1}$)», where *type* is the particular distribution function to be used and $v_0, \ldots, v_{n-1}$ are optional parameters that might be required by the probability distribution function. The notation of the "pdf" stereotype is illustrated in Fig. 4(e). Note that the use of the "pdf" stereotype is only meaningful for numerical alleles.

C. Modeling Example

We illustrate the use of ECML for specifying genotype models for a particular problem by using the knapsack problem as example. The genotype structure of the knapsack problem is rather simple. Items are represented by corresponding *Item* genes, of which each of them can be expressed by either a *yes* or *no* allele. Symbolic alleles have been chosen because selecting items is a discrete decision which does not require numerical encoding. Each of the possible outcomes of this decision is represented by a unique symbolic allele. The choice of symbols is irrelevant as long as the decoding mechanism is able to distinguish between them. A single association relationship R_{items} is used to connect the root element with the *Item* gene. This relationship allows us to specify the multiplicity m, i.e., the number of items considered for a particular problem instance. The general genotype model $M_{KS,G}$ for the knapsack problem is illustrated in Fig. 5.

In addition to specifying the structure and encoding of genotypes, it is necessary to add meta information regarding the processing aspects. For example, we need to specify information regarding the various variation operators. In case of the knapsack problem, a standard operator such as the one-point crossover, indicated by «var_{rc}:onepoint(0.6)», works well. The probability that this operator is applied when processing genotypes is $p_{var} = 0.6$. The operator does not require any additional parameters. Similarly, we specify that a random mutation operator is applied with a probability of $p_{var} = 1.0$ using the «var_{mt}:rand(1.0, $1/m$)» stereotype. This mutation operator has an additional parameter $v_0 = 1/m$ that indicates the mutation rate used by the operator.

Domain knowledge can be used to improve the performance for solving particular problem instances. For this purpose, it is sometimes useful to bias the selection of alleles during generation of individuals or performing mutation. For example, Khuri *et al.* bias their genetic algorithm to a particular problem instance by specifying a probability of 5% that an item is selected [22]. By doing so, they are able to achieve better results as compared to an unbiased version of their genetic algorithm. Such a bias can be specified in the genotype model by using the "ap" stereotype. The specialized genotype model $M_{KS,S}$, illustrated in Fig. 6, reflects this improvement.

IV. A Meta Evolutionary Algorithm: MEAα

Meta evolutionary algorithms (MEAs) refer to a family of EAs that is capable of interpreting high-level specifications (i.e., genotype models presented in ECML) and behaving accordingly. The behavior of a MEA can thus change during runtime if the information provided by the genotype model changes. This includes changes regarding operators (e.g., recombination, mutation, and local search) as well as changes in the genotype structure or its encoding. Different problems require different operators. Therefore, a MEA has to be equipped with a repository of supported operators. For example, EA libraries such as the ones mentioned above can be used for this purpose. During runtime, corresponding operators are selected on demand. Selection of operators is based on the meta information provided by the genotype model, which specifies what operators can be applied to a particular part of the genotype.

We have developed MEAα, a proof-of-concept Java implementation of MEA. MEAα is designed to be an interpreter that adapts its behavior dynamically depending on the meta information provided by the genotype model. While the schematic outline of MEAα follows that of standard EAs, its exact behavior depends entirely on the information specified in the genotype model. For example, whether or not a particular local search operator is applied depends on whether the corresponding "ls" stereotype is specified in the genotype model. Similarly, what kind of variation operation is applied to a particular part of the genotype depends on the corresponding "var" stereotype associated with a particular relationship.

MEAα is designed to be a solver which can be applied to any problem for which there is a suitable genotype model available. However, the algorithm itself cannot evaluate the individuals of a population (i.e., genotype instances) regarding their fitness. Neither can it compare two genotype instances and determine which one is better. These functions are inherently application-specific. We therefore assume that for every problem there is an application-specific evaluation function $f_E : g \rightarrow y$ which maps a genotype instance g to a performance value y. In addition, we assume that there is an application-specific comparison function $f_C : y_i, y_j \rightarrow \{true, false\}$ which indicates whether a performance value y_i is better than or equal to another performance value y_j. An outline of MEAα is illustrated in Fig. 7.

The binary relation $\preceq$ is used to determine the partial order of individuals in the search space X based on their associated performances by using the comparison function f_C. It holds that $g_i \preceq g_j$ if and only if $f_C(y_i, y_j) = true$ (i.e., if performance y_i is better than or equal to performance y_j). It is assumed

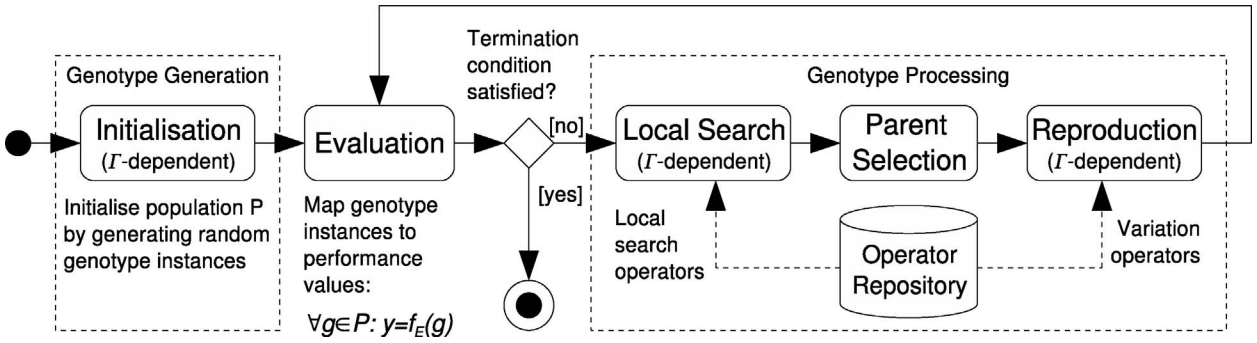


Fig. 7. Outline of MEAα. Corresponding local search and variation operators are provided by a repository and chosen during runtime. Initialization, local search, and reproduction make use of information provided by a genotype model Γ.

that for each problem, there is a non-empty set of optimal individuals $X_O \subset X$ for which the following holds:

$$\forall g_i, g_j \in X_O : g_i \preceq g_j \wedge g_j \preceq g_i \tag{1}$$

$$\forall g_i \in X_O, g_j \in X \backslash X_O : g_i \prec g_j. \tag{2}$$

This means that all individuals in X_O are equal in terms of their performance (1) and the individuals in X_O are better than the remaining individuals in X (2). The objective of the search performed by MEAα is to find X_O.

Some aspects of an EA are not covered by ECML in its current form. For example, parent selection and termination conditions are not specified by ECML. For our proof-of-concept implementation MEAα we make use of a tournament operator to perform parent selection. In addition, a fixed number of generations are used as termination condition. The functions f_E and f_C are provided externally in the form of problem-specific classes that implement corresponding interfaces. MEAα does not impose any restrictions or requirements regarding how fitness evaluation is actually done. This can be done by solving mathematical equations, using discrete event simulation, or any other process that is appropriate for the particular problem at hand. Later, in Section V, we will describe application examples and explain f_E and f_C used for these applications.

A. Genotype Generation

The genotype instances generated by MEAα are hierarchically structured as a direct result of the one-to-many relationships in the genotype model, i.e., genotype instances are tree structures. The various entities and relationships in the genotype model are reflected by nodes and links in genotype instances, respectively.

The initial population is created by generating a number of random genotype instances. A random genotype instance is created by parsing the genotype model and creating a corresponding number of nodes for each entity that is encountered. The exact number of nodes that are created depends on the multiplicity m of the corresponding association or activation relationship. Each of the created nodes is child nodes that are connected to their parent node by a corresponding link set. Parent and child nodes are nodes that correspond to the entities indicated on the left-hand side (from entity) and the right-hand side (to entity) of a one-to-many association/activation relationship, respectively. Each child node will be assigned a node index $0 \ldots m-1$ which allows to identify nodes that are connected by the same link set.

Parsing of the genotype model is done in a depth-first fashion by following outgoing association and activation relationships of all entities in the genotype model, starting with the root entity. Once an entity instance is created, all outgoing relationships and connected entities are processed in a recursive fashion. If a gene is encountered, then a random allele is selected from the set of possible alleles (determined by considering the various expression relationships for that particular gene). A uniform allele probability distribution is assumed unless specified otherwise by using the "ap" stereotype. In case of a numerical allele, a specific value is generated with respect to the given interval and "pdf" stereotype (if applicable).

For presentation purposes, we use a tree representation of genotype instances. Nodes that reflect gene entities are represented by rounded rectangles and named by the allele and the name of the gene, separated by a colon (i.e., *allele_name*:*gene_name*). For numerical alleles the actual value of the allele is included as well (i.e., *allele_name* = *value*:*gene_name*). Nodes that reflect cluster entities are represented by circles named after the cluster. An exception are nodes reflecting root entities which are named by the name of the genotype instance. Each node (except root nodes) has a node index assigned to it. In our notation of nodes, we indicate the node index for nodes that reflect genes or clusters in the upper right corner or at the top, respectively. Link sets are named according to the relationship they reflect: $L_{name,i}$, where *name* is the name of the corresponding relationship in the genotype model and *i* the index of the link set in case there are multiple link sets that correspond to the same relationship.

For example, consider the genotype model M_1 illustrated in Fig. 3(a). A random genotype instance is created by starting with the root entity M_1. A corresponding root node which reflects the name of the genotype instance is created. Association relationship R_1 connects root entity M_1 with gene entity G_1. In addition, R_1 has a multiplicity of 3. Therefore, three nodes are created by selecting random alleles (i.e., either *a* or *b*). These nodes are connected to the root node by a set of links $L_{1,0}$. Entity G_1 has no outgoing association relationships. If the selected allele does not have any outgoing activation relationships either, then no further processing has

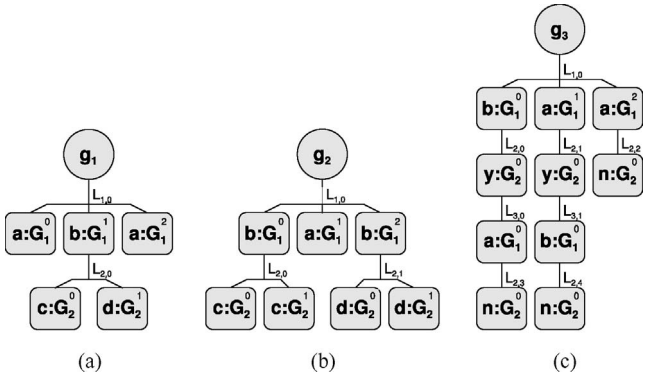


Fig. 8. Examples of possible genotype instances (a) g_1 and (b) g_2 for genotype model M_1 and a possible genotype instance (c) g_3 for genotype model M_2.

to be done. However, in case of a *b* allele, there is one outgoing activation relationship (i.e., R_2). This relationship is processed in the same recursive way as just described for relationship R_1. This process continues until the genotype of the individual is complete, i.e., if all entities and relationships in the genotype model have been processed. Fig. 8 illustrates examples of valid genotype instances for the genotype models M_1 and M_2, illustrated in Fig. 3.

B. Genotype Processing

Genotype processing in MEAα is concerned with performing local search and reproduction by means of corresponding operators. The meta information provided by the genotype model determines which operator is used for processing genotype instances. More precisely, the various "ls" and "var" stereotypes are used for this purpose. Operators are selected with a certain probability, specified by the corresponding stereotype (i.e., p_{ls} and p_{var}). As explained above, different operators can be applied to different parts of the genotype. In general, MEAα performs recombination by sub-tree swapping. The multiplicity of an association or activation relationship indicates that there are a number of corresponding nodes in the genotype instances. If recombination is performed, then these nodes represent the root nodes of the sub-tree structures that are subject for swapping. The exact way in which sub-tree structures are swapped between two parent genotype instance depends on the recombination operator.

For example, consider genotype model M_1 and the two possible genotype instances g_1 and g_2. Assume that a one-point crossover operation can be performed along relationship R_1. Since this relationship has a multiplicity of $m = 3$, there are three nodes, each reflecting an instance of a G_1 gene, in the genotype instance. These nodes represent the root nodes of the sub-tree structures used for swapping. Since a one-point crossover operation is performed, a crossover point $0 < k < m$ is randomly chosen. This means that all sub-tree structures with root nodes that have an index $j \geq k$ will be swapped between the two parent genotype instances g_1 and g_2. Fig. 9 illustrates how MEAα performs recombination for this example.

Mutation is performed in a similar fashion as recombination except that genotype instances are processed by completely replacing randomly selected sub-trees rather than by swapping them. For this purpose, randomly selected nodes (and all their child nodes) are removed from the genotype instance and replaced by new nodes. This node and all its child nodes (if any) are generated in the same fashion as discussed above in the context of genotype generation.

V. Application Examples

MEAα can be applied to problems across different domains and branches of evolutionary computing. We demonstrate this by applying our approach to the following three problems.

1) Job-shop scheduling problem: a well-known standard optimization problem. In this example, we show how different representations of the same problem can be described in ECML. In addition, we show how cluster entities can be used to structure the genotype and how ECML can be used to specify constraints.
2) Dynamic radiation detection problem: a problem with a real-world background where domain knowledge is changing over time. In this example, we demonstrate how the behavior of MEAα can adapt to changing domain knowledge by performing genotype model updates. In addition, we show how mixed coding genotypes can be modeled in ECML.
3) Grammatical evolution and genetic programming problem: we show how our approach can be applied to grammatical evolution and genetic programming. In this example, we discuss the general genotype model for genetic programming and explain how grammars in grammatical evolution can be modeled in ECML. We apply our approach to a symbolic regression problem and show that a more complex genotype model can lead to improved performance.

A. Job-Shop Scheduling Problem

The job-shop scheduling problem (JSP) is a well-known problem, where *n* jobs have to be scheduled on *m* machines. Each job has to be processed sequentially by each machine in a particular order, i.e., there are a total of *m* operations which have to be performed on each job. The *k*th operation of the *i*th job has a processing time $tp_{i,k}$. The problem is to find an optimal schedule for which the makespan, i.e., the total time to process all jobs, is minimized. Two constraints have to be considered. First, operations have to be performed in a specific sequence, i.e., a job has to be scheduled on all machines in a specific order. This is also referred to as the precedence constraint which states that a particular operation can only be performed if all preceding operations have already been performed. Second, a machine can only process one job at a time. Formally, the JSP can be stated as

$$\text{minimize} \quad C(g) = tf_{i,m-1} \quad \nexists j : tf_{j,m-1} > tf_{i,m-1} \tag{3}$$

$$\text{subject to} \quad \forall i, k : tf_{i,k-1} < ts_{i,k} \tag{4}$$

$$\text{and} \quad \forall 0 \leq l < m : max(a_l(t)) = 1 \quad t \geq 0 \tag{5}$$

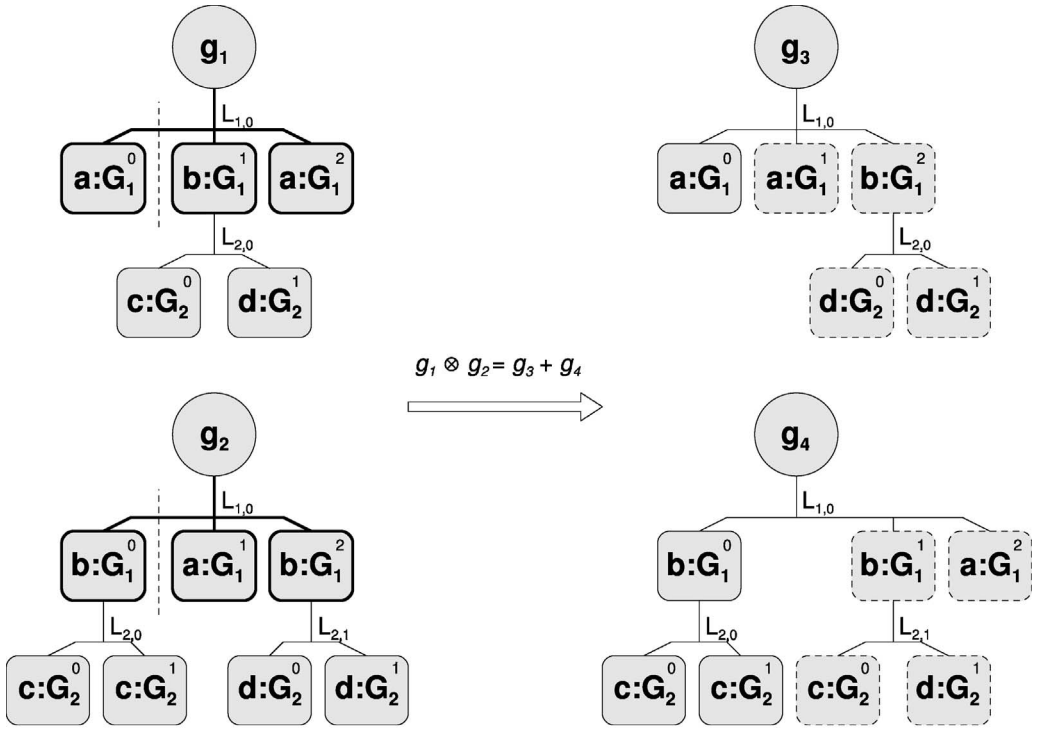


Fig. 9. Example for recombination of genotype instances g_1 and g_2 by performing sub-tree swapping. In this example, it is assumed that a one-point crossover operation is performed along relationship R_1. The corresponding link sets $L_{1,0}$ for relationship R_1 and the root nodes of the sub-tree structures are highlighted. A crossover point $k = 1$ is indicated by a dashed line that separates nodes with index $j \geq k$ from the node with index $j = 0$. In addition, the sub-tree structures that have been swapped during recombination are indicated by dashed lines in the resulting genotype instance g_3 and g_4.

where $C(g)$ is the makespan of a schedule obtained from a genotype g and $ts_{i,k}$ and $tf_{i,k} = ts_{i,k} + tp_{i,k}$ are the starting and finishing times of the kth operation of the ith job. In addition, $a_l(t)$ indicates the number of jobs that are processed by the lth machine at a given time t. The first constraint is expressed in (4) which asserts that the starting time $ts_{i,k}$ of the kth operation of the ith job is after the finishing time of the previous operation of the same job. The second constraint is expressed in (5) which asserts that for any machine l, there is at most one job processed at any given time $t \geq 0$.

There are various possible representations for the JSP. In general, there are two approaches to model the problem. It is distinguished between a direct representation and an indirect representation, depending on whether the schedule itself is represented by a solution [23]. If the schedule is not directly represented in the genotype, then it is necessary to decode the genotype in order to obtain the actual schedule. For example, an indirect representation may only specify the order in which jobs are being processed on a particular machine. However, the exact starting and finishing times of jobs, i.e., the schedule, is not directly specified. In addition, indirect representation may be ambiguous as there might be multiple feasible schedules which can be derived from it. Nevertheless, indirect representations are sometimes preferred over direct representations because of their simplicity. Here, we describe and compare two possible indirect representations for the JSP.

1) *Representation 1:* The first JSP representation and its corresponding decoding scheme follow the one discussed in [24]. A rank, ranging from $1 \ldots m$, is assigned to the m operations of each of the n jobs. This rank is unique among the operations of a single job, i.e., there can be only one operation within a job which is assigned a specific rank. Operations are decoded into schedules based on their ranks. Lower ranking operations are preferred over higher ranking operations and scheduled first. An operation is scheduled so that none of the constraints is violated. Depending on how ranks are assigned to operations, the resulting schedule is different. The problem is thus to find the optimal assignment of ranks to operations so that the makespan C is minimized.

The genotype model is partitioned into n job clusters. Each job in the genotype is associated with a sequence of m ranks, i.e., for each job there are m rank alleles. Each rank allele is associated with a particular operation. For example, the kth allele in the sequence encodes the rank of the kth operation. Within the context of a job, each rank must appear exactly once, i.e., a particular allele must appear only once within a job cluster. For this purpose, we introduce the "limited" constraint $\{\texttt{limited}(lb, ub)\}$. This constraint indicates how often a particular allele may appear within the scope of a cluster. For example, in the context of this JSP genotype representation, a $\{\texttt{limited(1, 1)}\}$ constraint is assigned to the various rank alleles, indicating that a particular rank allele must appear exactly once within the scope of a job cluster. However, since there are n job clusters, the same rank appears exactly n times within the genotype. Fig. 10 illustrates the genotype model M_{JSP1} and example genotype instance g_5 of the first JSP representation.

In addition to structural information, domain knowledge is reflected by the different kinds of crossover operators used for the two relationships in the genotype model. The order

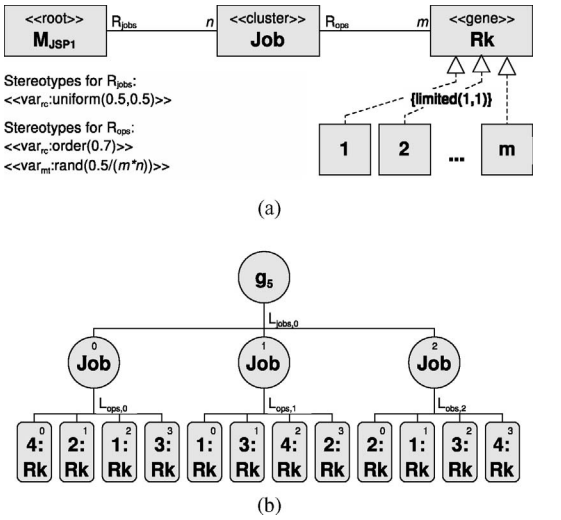


Fig. 10. (a) Genotype model M_{JSP1} and (b) a possible genotype instance g_5 for $n = 3$ jobs and $m = 4$ machines.

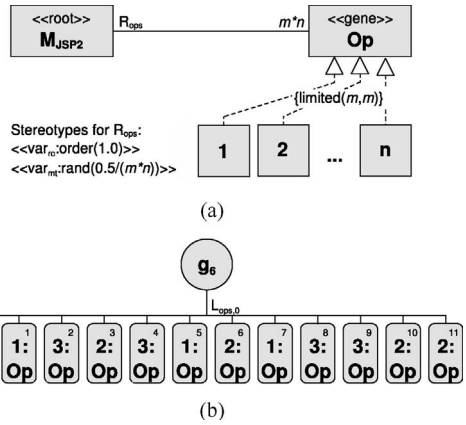


Fig. 11. (a) Genotype model M_{JSP2} and (b) a possible genotype instance g_6 for $n = 3$ jobs and $m = 4$ machines.

in which the operations of a particular job are ranked is important. Therefore, a corresponding order crossover [25] is used: «$\texttt{var}_{\texttt{rc}}$:order($p_{var}$)». This operator transmits information about the relative order of alleles when recombining segments from two different parents. While preserving the order of alleles within job clusters is important, this is not the case for job clusters. Therefore, a common uniform crossover operation can be applied: «$\texttt{var}_{\texttt{rc}}$:uniform($p_{var}$, v_0)», where parameter v_0 is the probability that a crossover operation is performed for a particular gene. The order and uniform crossover operations are associated with relationships R_{ops} and R_{jobs}, respectively. In addition, a mutation operator is associated with relationship R_{ops}: «$\texttt{var}_{\texttt{mt}}$:rand($p_{var}$, v_0)», where v_0 is the probability that a particular gene is randomly mutated. The probabilities for the various operators indicated in the genotype model have been chosen based on experimental experience.

2) *Representation 2:* The second JSP representation has originally been introduced by Bierwirth [26]. In this representation, a solution is encoded as an unpartitioned string in which each gene represents a job. Each job appears exactly m times which is ensured by using a {limited(m, m)} constraint. By scanning a genotype instance from left to right, the kth appearance of job j refers to the kth operation of this job. The order in which operations are performed is important. For this purpose, an order crossover operation is used along relationship R_{ops}. The probabilities for the various operators indicated in the genotype model have been chosen based on experimental experience. Like the previous JSP representation, this representation is an indirect representation and has to be decoded into a schedule. The advantage of this representation is that it never violates the precedence constraint. Fig. 11 illustrates the corresponding genotype model M_{JSP2} and example genotype instance g_6 of the second JSP representation.

3) *Evaluation Function f_E and Comparison Function f_C:* A genotype g is decoded into a valid schedule and used to determine the makespan $C(g)$. The actual decoding process depends on the representation used. For the two representations the corresponding decoding processes have already been explained in [24] and [26], respectively. The result y of the evaluation function $f_E(g)$, i.e., the performance of a genotype g, is

$$y = C(g). \tag{6}$$

Since the objective is to minimize the makespan, comparison of two JSP genotypes is simple. A genotype g_a is better than another genotype g_b if it results in a schedule that has a shorter makespan

$$g_a \preceq g_b \iff C(g_a) \leq C(g_b). \tag{7}$$

4) *Experimental Evaluation:* Evaluation is concerned with the comparison of the two representations. For this purpose, a standard problem instance from the literature has been used ("la32" [27]). A population size of 100 genotypes has been used and the evolutionary process is limited to 500 generations. We use the ability of MEAα to find the known optimum makespan C^{opt} over a set of 50 runs as performance metric. For the kth run the performance is measured in terms of relative deviation of the best makespan C_k^{best} found to the known optimum makespan C^{opt}

$$\delta_k = \frac{C_k^{best} - C^{opt}}{C^{opt}}. \tag{8}$$

For comparison, we consider the mean deviation δ over the entire set of experiments. Fig. 12 illustrates the convergence behavior for both representations, using genotype model M_{JSP1} and M_{JSP2}.

Both representations are indirect representations and need to be decoded into a schedule before the makespan can be determined. The decoding process can influence the performance. ECML is a flexible approach and allows designers to specify different genotype structures that can be used in combination with different decoding methods, i.e., a designer is not limited to a specific pre-defined representation but free to specify a representation that suits the problem best. In our example, we have described two possible genotype models for the JSP. Our results indicate that the more complex structure specified by genotype model M_{JSP1} is more suitable than M_{JSP2}.

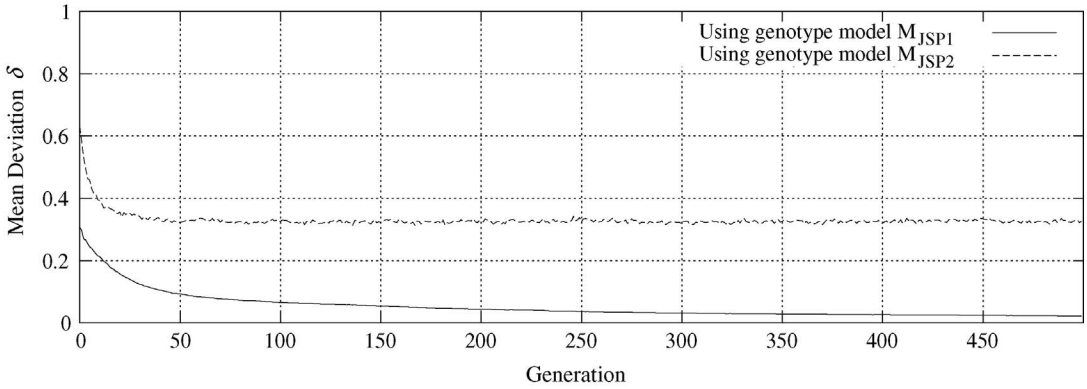


Fig. 12. Convergence behavior in terms of mean deviation δ using genotype models M_{JSP1} and M_{JSP2}.

B. Dynamic Radiation Detection Problem

A dynamic model identification problem in the context of a radiation detection application is considered. This application problem is concerned with the detection of a moving radiological dispersal device by static sensors. It has been originally discussed in [28]. Here, we briefly introduce the optimization problem and focus on the representation issues rather than on the application itself.

Detection devices, such as Geiger counters, can be used to measure the intensity of radioactivity at a particular location. However, this information is not enough to pinpoint a radiation source or classify its characteristics (i.e., type of radioactive substance and amount). A model identification approach can be used for classifying and localizing a radiation source. This approach is based on a model which accurately reflects the radiation absorbing properties of the environment and is used to estimate radiation intensities at any given location.

An estimation depends on a hypothesis regarding the assumed radiation source and its location, i.e., a hypothesis is concerned with the type of radiation substance j, its amount (i.e., total radioactivity) a, and its location pos_x, pos_y in a 2-D space. The radiation intensity at a specific location i can be estimated, based on a model RAM_j which accurately reflects the radiation absorbing properties depending on the type of radioactive substance

$$\hat{I}_i = RAM_j(a, pos_x, pos_y, i). \tag{9}$$

Different hypotheses are evaluated (assuming different values for j, a, pos_x, and pos_y) in order to estimate the radiation intensity $\hat{I}_i$ at reference location i. Sensor data is used to compare this estimated radiation intensity with the measured radiation intensity $\tilde{I}_i$ at the same reference location. Estimated and measured values from k reference locations are considered. The objective of the radiation detection problem is to find the optimal parameters of the model so that the error ϵ between the estimated and measured intensity values is minimized

$$\epsilon = \sum_{i=1}^{k} \frac{|\tilde{I}_i - \hat{I}_i|}{\tilde{I}_i}. \tag{10}$$

In a real-world application this optimization problem would have to be solved under real-time conditions, where the radiation source is expected to move. Therefore, the optimization problem has to be solved repeatedly in order to keep track of the current location of the radiation source. It is possible to simplify the optimization problem by significantly reducing the size of the search space using *a priori* knowledge about the radiation source—once available. We consider a multistage detection approach. At each stage, different kind of domain knowledge is available and can be incorporated in order to improve the optimization process. This has to be done dynamically as the domain knowledge becomes available.

In the first stage, no *a priori* information regarding the radiation source and its location is available. The goal in the first stage is to determine the characteristics (i.e., the isotope and the radioactivity) of the radiation source. Once this information is available, the optimization process of the second stage can make use of this information and focus on localization of the radiation source as it moves. For the second stage it is possible to further distinguish between determination of the initial location and the continuous tracking of the radiation source. For the initial localization it is necessary to consider the entire environment as a potential location. However, once the location has been determined, only a small area has to be considered in order to track the movement. The size of this area depends on the speed of the movement.

1) *Representation for the Classification Step:* Initially there is no *a priori* knowledge available, i.e., neither the characteristics nor the location of the radiation source is known. Therefore, all possible isotopes, radioactivity levels, and locations have to be considered, i.e., the entire search space. The genotype model $M_{RDP,G}$ for this general case is illustrated in Fig. 13.

The genotype model for the RDP imposes a genotype structure consisting of four genes. Four possible kinds of radioactive isotopes are considered in this example: Caesium-137, Cobalt-60, Iridium-192, and Radium-226. Each of them is represented as corresponding allele in the genotype model. Another gene encodes the radioactivity of the radiation source as an integer value ranging from 10 to 10 000 GBq (Bq = Becquerel) with a resolution of 1 GBq. Two genes are used to encode the x and y positions of the radiation source as integer values ranging from 0 to 199 units with a resolution

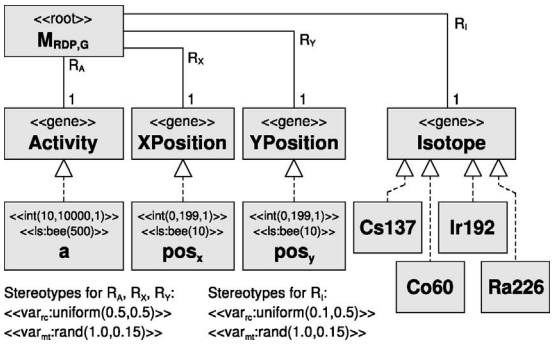


Fig. 13. RDP genotype model for the general case using no *a priori* information about the radiation source.

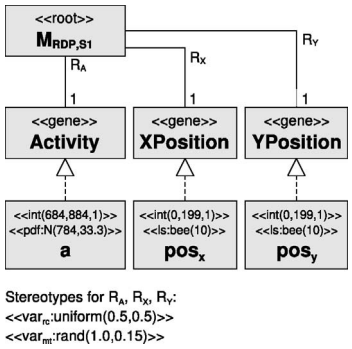


Fig. 14. Specialized genotype model $M_{RDP,S1}$ for the case that *a priori* information regarding the radiation substance and the radioactivity is known.

of 1 unit. This results in a search space with a size of $4 \times 9991 \times 200 \times 200 \approx 1.6 \times 10^9$ solutions.

In this example, the bee algorithm [29] is used to perform local search for a, pos_x, pos_y alleles that encode the radioactivity and the position of the radiation source. The bee algorithm needs to know the size of the neighborhood in order to perform a local search. The neighborhood size is indicated by a parameter. For example, the stereotype «`ls:bee(500)`» attached to the allele a indicates that a bee algorithm with a neighborhood size of 500 GBq is used.

2) *Representation for the Localization Step:* Once the radiation substance and the amount of its radioactivity (i.e., j and a) have been determined, the detection process only needs to be concerned with the location. Localization is concerned with solving the same optimization problem. However, for localization it is possible to re-use knowledge from the preceding classification step. Since the type of radioactive substance and its radioactivity has been determined in the preceding step, it is possible to re-use this *a priori* information for subsequent tracking of the radiation source. This domain knowledge can be incorporated into the genotype model accordingly, i.e., the isotope can be removed and the activity range can be limited to a small range. The first specialized genotype model $M_{RDP,S1}$ for this optimization scenario is illustrated in Fig. 14.

The first detection stage using genotype model $M_{RDP,G}$ considers the entire search space and uses local search. As mentioned above, real-time constraints have to be considered. In order to track a moving radiation source, the problem has to be solved in a timely manner. This is achieved by reducing

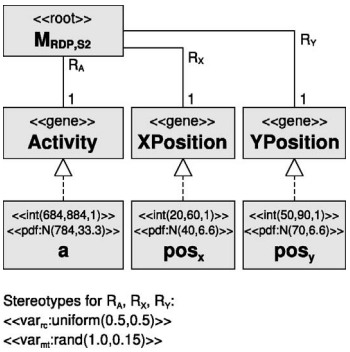


Fig. 15. Specialized genotype models $M_{RDP,S2}$ for the case that *a priori* information regarding the radiation substance, the radioactivity, and the approximate location of the source is known.

the size of the search space. Genotype model $M_{RDP,S1}$ is incorporating knowledge about the radiation source which is available after the first detection stage. As illustrated in Fig. 14, genotype model $M_{RDP,S1}$ reflects the case where $a = 784$ (the isotope has been determined previously and is thus removed from the genotype model). The resulting search space has a size of $201 \times 200 \times 200 \approx 8 \times 10^6$ solutions. Compared with the search space of the general case, this is much smaller. In addition, local search is no longer applied to allele a. Instead, a normal distribution function with $\mu = 784$ and $\sigma = 33.3$ is assumed using a corresponding «`pdf:N(784, 33.3)`» stereotype.

An even more compact search space can be obtained once the location of the radiation source has been initially approximated. Since the speed of the source is limited, subsequent localization cycles only need to consider a relatively small search area. The second specialized genotype model $M_{RDP,S2}$, which is illustrated in Fig. 15, incorporates the same *a priori* knowledge as the first specialized genotype model $M_{RDP,S1}$. In addition, it incorporates knowledge regarding the approximate location of the radiation source. Genotype model RDP_{S2}, as illustrated in Fig. 15, reflects the case where a position of the radiation source is $x = 40$ and $y = 70$. The resulting search space has a size of $1 \times 201 \times 41 \times 41 = 337881$ solutions. In addition, local search is no longer applied to alleles pos_X and pos_Y. Instead, corresponding normal distribution functions are assumed using the «`pdf:N(40, 6.6)`» and «`pdf:N(70, 6.6)`» stereotype, respectively.

3) *Evaluation Function f_E and Comparison Function f_C:* Evaluation is done by solving the radiation model in order to obtain the estimated radiation intensities and calculating the error ϵ between the estimated and measured intensities. The result y of the evaluation function $f_E(g)$, i.e., the performance of a genotype g, is

$$y = \epsilon. \tag{11}$$

The evaluation function f_E is time-dependent because the measured intensities depend on the position of the radiation source, which is assumed to move.

The objective is to minimize the error ϵ between the estimated and measured radiation intensities

$$g_i \preceq g_j \iff \epsilon_i \leq \epsilon_j. \tag{12}$$

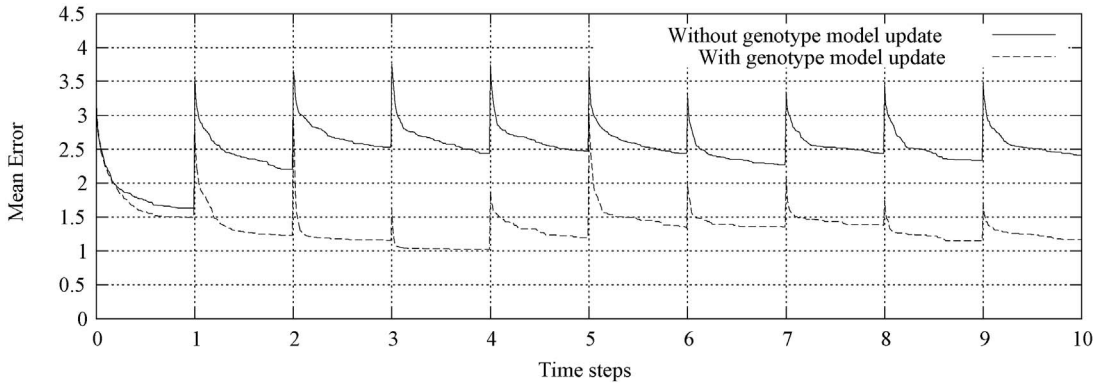


Fig. 16. Convergence behavior in terms of mean error with and without genotype model update. Each timestep represents 100 generations. The evolutionary process is re-initialized after each timestep.

4) *Experimental Evaluation:* For evaluation purposes, we consider the convergence behavior for the two cases where genotype model update is used and not used. In case no model update is used, the entire search space has to be considered for all detection stages, i.e., the genotype model for the general case is used throughout the experiment. A set of 50 experiments has been performed. Each experiment covers 10 timesteps during which the radiation source is changing its position. For each timestep, the evolutionary process is re-initialized and runs for 100 generations. Classification and initial localization is performed during the first and second timestep, respectively. All subsequent timesteps are concerned with localization. Population sizes of 200, 100, and 50 genotypes are used for classification, initial localization, and all subsequent localization timesteps, respectively. Fig. 16 illustrates the convergence behavior in terms of the mean error for two cases: with and without genotype model update.

Without dynamic model update, the entire search space has to be considered by the search process. In contrast, dynamic model update allows restricting the search space by using available information about the radiation source. As a consequence, the performance of MEAα is better when performing dynamic model updates. This example demonstrates the flexibility of our approach. A MEA can dynamically adapt its behavior if the genotype model changes. For example, genotype model $M_{RDP,G}$ specifies the use of a local search method. Once classification and initial localization is performed, local search is no longer required and thus not specified in genotype model $M_{RDP,S2}$. Consequently, MEAα performs local search operations only if genotype models $M_{RDP,G}$ and $M_{RDP,S1}$ are used. In addition, the structure of the genotype is changing as well. None of the specialized genotype models $M_{RDP,S1}$ and $M_{RDP,S2}$ considers an isotope gene.

C. Genetic Programming and Grammatical Evolution Problems

Genetic programming is considered as a branch of evolutionary computing, next to genetic algorithms, evolution strategies, and evolutionary programming, which distinguishes itself from other branches by using tree structures as chromosomes. Furthermore, Eiben and Smith [30] explained that genetic algorithms are typically applied to optimization problems as compared to genetic programming which is typically used for model identification. In genetic programming, problems can be generally defined by a set of function symbols (i.e., $F = \{F_0, F_1, \ldots, F_{n_F-1}\}$) and a set of constant symbols (i.e., $C = \{C_0, C_1, \ldots, C_{n_C-1}\}$).[4] Each function takes at least one parameter. The number of parameters of the kth function F_k is m_k, where $m_k > 0$. The general genotype model for genetic programming is illustrated in Fig. 17.

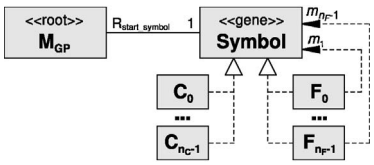


Fig. 17. General genotype model of a genetic programming problem.

Grammatical evolution is very similar to genetic programming. Problems are specified by grammars given in Backus-Naur form (BNF). Any syntactically correct program represents a solution. A BNF grammar can be represented by the tuple $\{N, T, P, S\}$, where N is the set of non-terminal symbols, T the set of terminal symbols, P the set of production rules, and $S \in N$ is a start symbol. Grammatical evolution is therefore slightly different from traditional genetic programming which only considers sets of constants and functions. While the genotype model structure in genetic programming is always the same (see Fig. 17), the genotype model structure for grammatical evolution is dictated by the production rules and is therefore different from problem to problem. However, it is possible to obtain the standard genetic programming genotype model structure by simplifying a grammatical evolution genotype model structure.

ECML is capable of specifying more complex genotype structures such as tree structures. It is therefore possible to use ECML to specify genetic programming and grammatical evolution problems. In order to show this, we take an example

[4]Note that in genetic programming the term "terminal symbol" is used instead of "constant symbol." However, the same term is also used in grammatical evolution but with a different meaning. Therefore, in order to avoid confusion, we stick to the convention to use "constant symbol" whenever we refer to what is usually known as "terminal symbol" in genetic programming.

problem from the literature which is concerned with symbolic regression.

1) *Grammatical Evolution Representation:* The objective in symbolic regression is to find a mathematical expression in symbolic form that maps given pairs of input and output values. For example, the target function considered by O'Neill and Ryan [8] is

$$f(u) = u^4 + u^3 + u^2 + u \tag{13}$$

where u is in the range $[-1, 1]$. The following grammar for this problem has been described and explained by O'Neill and Ryan [8]:

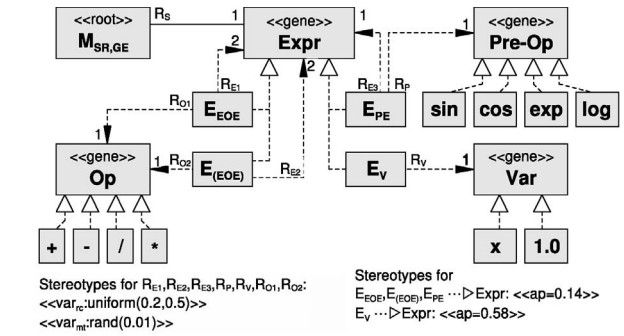
```
<expr> ::= <expr><op><expr>            (a.1)
         | (<expr><op><expr>)          (a.2)
         | <pre-op>(<expr>)            (a.3)
         | <var>                       (a.4)
<op> ::= + | - | / | *                 (b)
<pre-op> ::= sin | cos | exp | log      (c)
<var> ::= x | 1.0                      (d)
```

This grammar consists of four derivation rules (a)–(d) and can be translated into a genotype model as illustrated in Fig. 18(a). Each non-terminal symbol in the grammar is represented by a corresponding gene in the genotype model, while terminal symbols are represented by corresponding alleles. Four additional alleles (E_{EOE}, $E_{(EOE)}$, E_{PE}, E_V) are used to represent the four possible forms of an expression (a.1 ... a.4). Depending on the expression allele, there are corresponding activation relationships to other genes in the genotype model. For example, the form (a.1) involves two expressions and one operator. This is acknowledged in the genotype model by relationships R_{E1} and R_{O1} with multiplicities of 2 and 1, respectively.

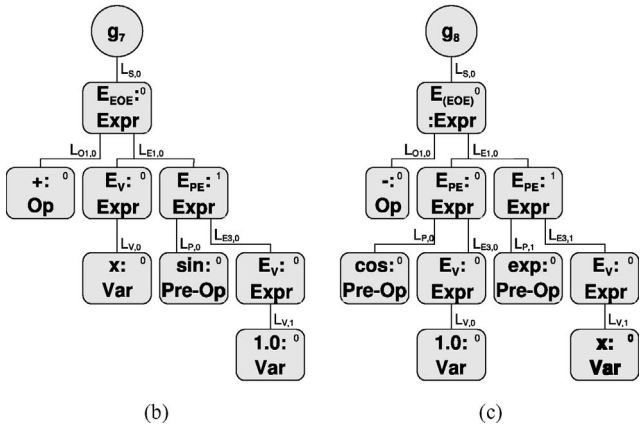
For example, consider genotype instances g_7 and g_8 in Fig. 18(b) and (c), respectively. By parsing the tree structure, starting from the root node, they can be decoded into $x + \sin(1.0)$ and $(\cos(1.0) - \exp(x))$, respectively. Notice the outer brackets in the second example as they are important. Derivation rules a.1 and a.2 are almost identical except for the brackets. When decoding larger solutions, this difference has to be acknowledged. Otherwise the equation $a*b+c = a*(b+c)$ would be correct, which is obviously not true for most cases. The genotype model distinguishes between these rules by using two different alleles E_{EOE} and $E_{(EOE)}$. Although these two alleles are otherwise identical, i.e., they have exactly the same kind of activation relationships, the decoding process has to handle them accordingly.

2) *Genetic Programming Representation:* Representations of grammatical evolution problems can be simplified into general genotype model structure for genetic programming. Simplification of the symbolic regression problem is done in two steps. First, the set of constant symbols has to be identified. Constants are characterized by not having any forthleading relationships. This includes most of the terminal symbols in the grammar: $+$, $-$, $/$, $*$, x, and 1.0. Not all terminal grammar symbols are constants, i.e., not all terminal grammar symbols are terminal in the sense of the meaning in the context of genetic programming. The terminal symbols sin, cos, exp, and log are not constant (i.e., not "terminal" in genetic programming) because there are forthleading activation relationships. In the second step, the set of function symbols have to be identified. The first group of function symbols consists of all unary functions, i.e., functions that take one parameter: sin, cos, exp, and log. The second group of function symbols consists of all binary functions, i.e., functions that take two parameters. When identifying the function symbols it is necessary to take the derivation rules into consideration: $+$, $-$, $/$, $*$, $(+)$, $(-)$, $(/)$, and $(*)$. The genotype model of the simplified symbolic regression model is illustrated in Fig. 19.

Note that in Fig. 19 we did not illustrate every single relationship but instead summarized them in order to keep the



(a)



(b) (c)

Fig. 18. (a) Genotype model $M_{SR,GE}$ of the symbolic regression problem based on the original set of production rules and (b), (c) genotype instances g_7 and g_8 for two valid solutions.

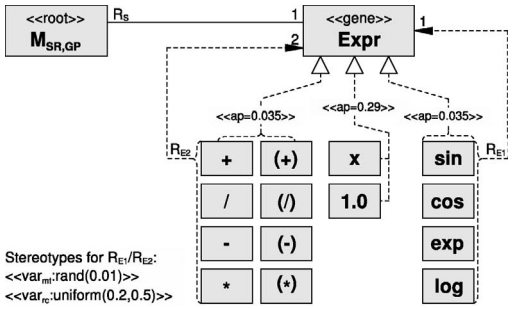


Fig. 19. Genotype model $M_{SR,GP}$ of the simplified symbolic regression problem based on the general genotype model structure for genetic programming.

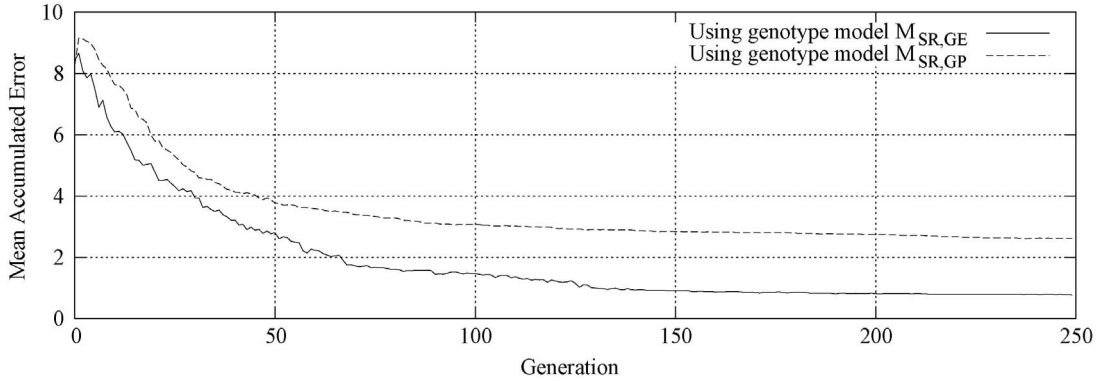


Fig. 20. Convergence behavior in terms of mean accumulated error for genotype models $M_{SR,GE}$ and $M_{SR,GP}$.

figure concise. The unary and binary function relationships are substituted by single relationships R_{E1} and R_{E2}, respectively. The full genotype model should have a single relationship from every allele that represents a function symbol to the expression gene (e.g., $\sin \rightarrow Expr$, $(+) \rightarrow Expr$, ...).

3) *Evaluation Function f_E and Comparison Function f_C:* Evaluation of a genotype g is done by decoding it into a candidate function $h(u)$ and calculating the accumulated error $E(g)$, which is the sum of all deviations of the candidate function to the reference function $f(u)$ over all test values $u \in U = \{-1.0, -0.9, \ldots, 0.9, 1.0\}$

$$E(g) = \sum_{u \in U} |f(u) - h(u)|. \tag{14}$$

The result y of the evaluation function f_E, i.e., the performance of a genotype g, is

$$y = E(g). \tag{15}$$

Two genotypes g_a and g_b are compared based on their accumulated errors

$$g_a \preceq g_b \iff E(g_a) \leq E(g_b). \tag{16}$$

4) *Experimental Evaluation:* For evaluation purposes, we consider the convergence behavior of MEAα for both genotype models discussed above: the grammatical evolution genotype model $M_{SR,GE}$ and the simplified genetic programming genotype model $M_{SR,GP}$. The various allele probabilities have been chosen to result in equivalent behavior of MEAα for both models. For example, in genotype model $M_{SR,GE}$ the allele probability for $E_V \rightarrow Expr$ is $p_a = 0.58$. If $Expr$ is expressed by E_V, then an additional Var gene is activated which can be expressed with equal likelihood by either the x or the 1.0 allele. For each $Expr$ gene in the genotype, there is therefore a probability of $0.58/2 = 0.29$ that allele x or 1.0 is selected. This is equivalent to the allele probabilities of $p_a = 0.29$ separately assigned to $x \rightarrow Expr$ and $1.0 \rightarrow Expr$ in genotype model $M_{SR,GP}$. A set of 50 runs has been performed for each genotype model using a population size of 250 genotypes and a fixed number of 250 generations. The mean accumulated error over the set of experiments has been calculated. Fig. 20 illustrates the convergence behavior in terms of mean accumulated error for both genotype models.

The results indicate that using the more complex $M_{SR,GE}$ genotype model is more suitable to solve the problem. Both representations used for the symbolic regression problem are direct representations, i.e., solution are directly encoded by the genotype. A decoding mechanism is therefore not required and cannot influence performance. The experimental results for this example indicate that a more complex structure, such as the one specified by $M_{SR,GE}$, can improve the performance compared to a less complex genotype structure.

VI. Discussion

De Jong argues that one of the most important issues that have to be addressed when applying an EA to a new application area is how to represent the individuals that have to be evolved [39]. He further explains that a good representation captures important features of the application domain which are necessary to effectively solve the problem. ECML is a high-level modeling language and allows the capturing of important features by providing means to model the objects of interest for a particular problem. In addition, ECML also allows to specify genotype structures of varying complexity. This can be useful in cases where a more complex structure can lead to performance improvements. For example, our experimental results presented in Section V-C indicate that using more complex structure improves the performance.

In comparison with existing approaches, ours is closest to grammar-based approaches. However, there are a number of differences. Grammars are used to map genotypes to a syntactically correct sentence which represent the phenotype. Although these phenotypes have a logical tree structure, this information is not preserved in genotypes. For example, grammatical evolution makes use of a binary string representation. In contrast, our approach preserves the logical tree structure in the genotype. In addition, grammar-based approaches have shortcomings with respect to numerical value encoding and crossover points. For this purpose, Nicolau and Dempsey introduced grammar-based extensions [31] that allow for specification of numerical values and crossover points in grammars. Although their approach works, it shows a general problem with grammars as they are originally used for language processing where there is no need for numerical values (i.e., numbers are strings of symbols) or recombination. In contrast, ECML explicitly distinguishes between symbolic and numerical alleles. In addition, relationships between the various entities of a genotype model are used as possible recombination points.

Existing approaches are concerned with compiling an EA specification in order to obtain an actual instance of the algorithm [5], [6], [10], [11]. An EA, which is generated in this way, contains the code for performing specific operations (e.g., uniform crossover) on genotypes with a specific structure and encoding (e.g., binary string). The disadvantage of a compiler approach is that EAs that have been generated in this way cannot dynamically adapt to changes in the specification, i.e., the specification is static and requires re-compilation. For example, making use of another kind of operator or changing the genotype structure or its encoding is not possible. Unlike static approaches, a MEA is capable of changing its behavior as the information provided by the genotype model is changing. This includes changes regarding the use of certain operators (e.g., recombination, mutation, and local search) as well as changes in the genotype structure.

Many real-world applications need to repeatedly solve a problem. For example, dynamic-data driven application systems [32] and symbiotic simulation systems [33], [34] are driven by real-time sensor data obtained from a physical system. Some of these systems employ optimization for operational decision making. In such a system, decision making depends on the current operating conditions. As these conditions may change over the course of time, problem-specific knowledge may change as well. Hard-coding domain knowledge is thus problematic as it may not be easily changed. For this kind of application, a dynamic way of incorporating domain knowledge is required. In Section V-B, we have discussed a dynamic optimization problem in the context of a radiation detection application and explained how MEAα adapts by updating the genotype model if domain knowledge is changing with time.

VII. Conclusion and Future Work

We have introduced ECML, a formal modeling language for evolutionary computing based on UML. This language can be used to specify genotype structures and their encoding as well as meta information regarding various processing aspects of an EA. In addition, we have introduced the concept of MEAs and described our proof-of-concept implementation of this concept. MEAα is capable of interpreting genotype models specified in ECML and process genotypes instances accordingly. With our approach we effectively separated domain knowledge from the EA implementation. As a consequence, MEAα is more flexible than existing EAs. We have demonstrated the applicability of our approach using various kinds of problems.

Our approach is novel in several ways. First, ECML is not a domain-specific modeling language and thus not restricted to a particular problem or branch of evolutionary computing. Instead, it is generally applicable. Second, the concept of MEAs involves interpretation of genotype models rather than compiling of a specification into an executable EA instance.

Although we have discussed the application of ECML in the context of genetic algorithms and genetic programming, this does not imply that ECML is limited to this kind of algorithm. For example, ECML could be applied to differential evolution [35] and particle swarm optimization [36] by using numerical alleles to encode individuals. Furthermore, in the context of compact genetic algorithms [37], ECML could be applied by using allele probability stereotypes to realize the probability vector needed to create individuals. We intend to investigate further some of the above-mentioned applications of ECML in our future work.

A common language for modeling genotypes could be a step closer to a unified approach in evolutionary computing. Other researchers have already made efforts toward this direction. For example, De Jong summarized the various branches of evolutionary computing and presented a more unified view of the field [38]. ECML represents another step toward unification. However, more work is needed to include EA components that are not yet supported by ECML. For example, parent selection and termination conditions are currently not included in ECML. Furthermore, a formal specification of the various operators that are used is missing. Also, well-known concepts, such as epistasis, are currently not supported by ECML. These issues are thus subject to future work.

References

[1] D. Wolpert and W. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, Apr. 1997.

[2] D. W. Corne and J. D. Knowles, "No free lunch and free leftovers theorems for multiobjective optimization problems," in *Proc. 2nd Int. Conf. Evol. Multi-Criterion Optimization*, 2003, pp. 327–341.

[3] D. Wolpert and W. Macready, "Coevolutionary free lunches," *IEEE Trans. Evol. Comput.*, vol. 9, no. 6, pp. 721–735, Dec. 2005.

[4] P. P. Bonissone, R. Subbu, N. Eklund, and T. R. Kiehl, "Evolutionary algorithms+domain knowledge = real-world evolutionary computation," *IEEE Trans. Evol. Comput.*, vol. 10, no. 3, pp. 256–280, Jun. 2006.

[5] M. Keijzer, J. Merelo, G. Romero, and M. Schoenauer, "Evolving objects: A general purpose evolutionary computation library," in *Proc. 5th Int. Conf. Artif. Evol.*, 2002, pp. 231–242.

[6] P. Collet, E. Lutton, M. Schoenauer, and J. Louchet, "Take it EASEA," in *Proc. 6th Int. Conf. Parallel Problem Solving Nature*, 2000, pp. 891–901.

[7] P. Whigham, "Grammatically-based genetic programming," in *Proc. Workshop Genet. Programming: From Theory to Real-World Applicat.*, 1995, pp. 33–41.

[8] M. O'Neill and C. Ryan, "Grammatical evolution," *IEEE Trans. Evol. Comput.*, vol. 5, no. 4, pp. 349–358, Aug. 2001.

[9] J.-Y. Jung and J. A. Reggia, "Evolutionary design of neural network architectures using a descriptive encoding language," *IEEE Trans. Evol. Comput.*, vol. 10, no. 6, pp. 676–688, Dec. 2006.

[10] C. Veenhuis, K. Franke, and M. Köppen, "A semantic model for evolutionary computation," in *Proc. 6th Int. Conf. Soft Comput.*, 2000.

[11] C. Veenhuis, M. Köppen, and K. Franke, "Hierarchical modeling of evolutionary computation," in *Soft Computing: Methodologies and Applications*, vol. 32. Berlin/Heidelberg, Germany: Springer, 2005, pp. 97–111.

[12] Object Management Group. (2009, Feb.). *OMG Unified Modeling Language (OMG UML)* [Online]. Available: http://www.omg.org/spec/UML/2.2/

[13] N. Radcliffe, "The algebra of genetic algorithms," *Ann. Math. Artif. Intell.*, vol. 10, no. 4, pp. 339–384, Dec. 1997.

[14] C. Mattiussi and D. Floreano, "Analog genetic encoding for the evolution of circuits and networks," *IEEE Trans. Evol. Comput.*, vol. 11, no. 5, pp. 596–607, Oct. 2007.

[15] R. Cleary and M. O'Neill, "An attribute grammar decoder for the 01 multiconstrained knapsack problem," in *Proc. 5th Eur. Conf. Evol. Comput. Combinatorial Optimization*, 2005, pp. 34–45.

[16] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3rd ed. Reading, MA: Addison-Wesley, 2004.

[17] Object Management Group. (2008, Nov.). *OMG Systems Modeling Language (OMG SysML)* [Online]. Available: http://www.sysml.org/specs.htm

[18] Object Management Group. (2006, May). *Object Constraint Language Specification* [Online]. Available: http://www.omg.org/spec/UML/2.2/

[19] H. Pearson, "Genetics: What is a gene?" *Nature*, vol. 441, no. 441, pp. 398–401, May 2006.
[20] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Berlin, Germany: Springer, 2004.
[21] Y. Ong, M. Lim, and X. Chen, "Research frontier: Memetic computation—past, present and future," *IEEE Comput. Intell. Mag.*, vol. 5, no. 2, pp. 24–31, May 2010.
[22] S. Khuri, T. Bäck, and J. Heitkötter, "The zero/one multiple knapsack problem and genetic algorithms," in *Proc. ACM Symp. Appl. Comput.*, 1994, pp. 188–193.
[23] M. Gen, Y. Tsujimura, and E. Kubota, "Solving job-shop scheduling problems by genetic algorithm," in *Proc. IEEE Int. Conf. Syst. Man Cybern.*, vol. 2. Oct. 1994, pp. 1577–1582.
[24] J. Garen. *Multiobjective Job-Shop Scheduling with Genetic Algorithms Using a New Representation and Standard Uniform Crossover* [Online]. Available: http://webhost.ua.ac.be/eume/workshops/momh/momh-10.pdf
[25] L. Davis, *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold, Jan. 1991.
[26] C. Bierwirth, "A generalized permutation approach to job shop scheduling with genetic algorithms," *OR Spectrum*, vol. 17, nos. 2–3, pp. 87–92, 1995.
[27] S. Lawrence, "Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques," Graduate School Ind. Admin., Carnegie-Mellon Univ., Pittsburgh, PA, 1984.
[28] H. Aydt, S. J. Turner, W. Cai, M. Y. H. Low, and R. Ayani, "Symbiotic simulation model validation for radiation detection applications," in *Proc. 23rd Workshop Principles Adv. Distributed Simul.*, 2009, pp. 11–18.
[29] D. Pham, A. Ghanbarzadeh, E. Koç, S. Otri, S. Rahim, and M. Zaidi, "The bees algorithm: A novel tool for complex optimization problems," in *Proc. 2nd I* PROMS Virtual Conf. Intell. Prod. Mach. Syst.*, 2006, pp. 454–459.
[30] A. Eiben and J. Smith, *Introduction to Evolutionary Computing*. Berlin, Germany: Springer, 2003.
[31] M. Nicolau and I. Dempsey, "Introducing grammar based extensions for grammatical evolution," in *Proc. IEEE Congr. Evol. Comput.*, 2006, pp. 648–655.
[32] National Science Foundation. (2005). DDDAS: Dynamic data driven applications systems. *Program Solicitation 05-570* [Online]. Available: http://www.nsf.gov/pubs/2005/nsf05570/nsf05570.htm
[33] R. Fujimoto, D. Lunceford, E. Page, and A. M. Uhrmacher, Eds., "Grand challenges for modeling and simulation: Dagstuhl report," Seminar 02351, Tech. Rep. 350, Aug. 2002.
[34] H. Aydt, S. J. Turner, W. Cai, and M. Y. H. Low, "Symbiotic simulation systems: An extended definition motivated by symbiosis in biology," in *Proc. 22nd Workshop Principles Adv. Distributed Simul.*, Schloss Dagstuhl, Wadern, Germany, 2008, pp. 109–116.
[35] R. Storn and K. Price, "Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.
[36] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, vol. 4. 1995, pp. 1942–1948.
[37] G. R. Harik, F. G. Lobo, and D. E. Goldberg, "The compact genetic algorithm," *IEEE Trans. Evol. Comput.*, vol. 3, no. 4, pp. 287–297, Nov. 1999.
[38] K. A. De Jong, *Evolutionary Computation: A Unified Approach*. Cambridge, MA: MIT Press, 2006, ch. 6.6, pp. 185–188.

Heiko Aydt (S'09) received the M.S. degree in information technology from the Royal Institute of Technology, Stockholm, Sweden. He is currently pursuing the Ph.D. degree from the Division of Computer Science, School of Computer Engineering, Nanyang Technological University, Singapore.

He is currently a Research Associate with the Division of Computer Science, School of Computer Engineering, Nanyang Technological University. His current research is concerned with automated problem solving using symbiotic simulation which involves simulation-based online optimization. His current research interests include multiagent systems and simulation, symbiotic simulation, evolutionary computing, and artificial intelligence.

Stephen John Turner (M'01) received the M.A. degree in mathematics and computer science from Cambridge University, Cambridge, U.K., and the M.S. and Ph.D. degrees in computer science from Manchester University, Manchester, U.K.

He is currently a Professor of computer science and the Head of the Computer Science Division, School of Computer Engineering, Nanyang Technological University, Singapore. His current research interests include parallel and distributed simulation, grid computing, high performance computing, and multiagent systems.

Dr. Turner is the Steering Committee Chair of the Principles of Advanced and Distributed Simulation Conference and is an Area Editor for the *ACM Transactions on Modeling and Computer Simulation*.

Wentong Cai (M'93) received the Ph.D. degree in computer science from the University of Exeter, Exeter, U.K., in 1991.

He is currently a Professor with the Division of Computer Science, School of Computer Engineering, Nanyang Technological University, Singapore. He is the Director of the Parallel and Distributed Computing Center, Nanyang Technological University, Singapore. His current research interests include modeling, simulation (particularly parallel and distributed simulation and agent-based simulation), parallel and distributed computing (particularly grid and cluster computing).

Dr. Cai is an Associate Editor of the *ACM Transactions on Modeling and Computer Simulation*, and is an editorial board member of the international journal *Multiagents and Grid Systems*.

Malcolm Yoke Hean Low (M'96) received the Bachelors and Masters of Applied Science degrees in computer engineering from Nanyang Technological University (NTU), Singapore, and the D.Phil. degree in computer science from Oxford University, Oxford, U.K.

He is currently an Assistant Professor with the School of Computer Engineering, NTU. Prior to this, he was with the Planning and Operations Management Group, Singapore Institute of Manufacturing Technology, Singapore. He is the Principal Investigator of the Singapore Ministry of Defense funded project on "Evolutionary computing methodologies for modeling, simulation and analysis" under the Defense Innovation Research Program, as well as the Principal Investigator in the Maritime and Port Authority of Singapore and APL funded research project on "Optimization of stowage plans for large containerships." He has extensive experience in modeling and simulation, planning and scheduling, as well as parallel computing. His current research interests include the application of parallel/distributed simulation, grid computing and agent technology for the modeling, simulation, analysis, and optimization of complex systems.

Yew-Soon Ong (M'03) received the B.S. and M.S. degrees in electrical and electronics engineering from Nanyang Technological University (NTU), Singapore, in 1998 and 1999, respectively, and the Ph.D. degree in artificial intelligence in complex design from the Computational Engineering and Design Center, University of Southampton, Southampton, U.K., in 2002.

He is currently an Associate Professor and Director of the Center for Computational Intelligence, School of Computer Engineering, NTU. His current research interests include computational intelligence spans across memetic computing, evolutionary design, optinformatics, machine learning, agent-based systems, and cloud computing.

Dr. Ong is serving as the Technical Editor-in-Chief of the *Memetic Computing Journal*, the Chief Editor of a book series on studies in adaptation, learning, and optimization, the Associate Editor of the IEEE COMPUTATIONAL INTELLIGENCE MAGAZINE, the IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS—PART B, the *International Journal of System Science*, and the *Soft Computing Journal*. He is the Chair of the Task Force on Memetic Computing in the IEEE Computational Intelligence Society Emer-

gent Technology Technical Committee, and has served as a Guest Editor of the IEEE COMPUTATIONAL INTELLIGENCE MAGAZINE, the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, the IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS—PART B, the IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS—PART C, the *Journal of Genetic Programming Evolvable Machine*, and the *Soft Computing Journal*.

Rassul Ayani (M'09) received the Dipl. Ing. degree from the University of Technology, Vienna, Austria, the M.S. degree from the University of Stockholm, Stockholm, Sweden, and the Ph.D. degree from the Royal Institute of Technology (KTH), Stockholm.

He is currently a Professor of Computer Science with the School of Information and Communication Technology, KTH. He has held visiting positions with several universities, including the Georgia Institute of Technology, Atlanta, the University of Waterloo, Waterloo, ON, Canada, the National University of Singapore, Singapore, and Nanyang Technological University, Singapore. He has been working on parallel and distributed systems for the past 20 years. During the 1990s, his research interests included mainly parallel and distributed simulation. His current research interests include semantic interoperability and composability of model components, symbiotic simulations, optimization of business process models, service composition, and context aware service discovery.

Dr. Ayani has served as a Program Chair and a program committee member of numerous international conferences, and has served as an Area/Associate Editor for seven international journals, including the *ACM Transactions on Modeling and Computer Simulation*. He has been a panel member of the European Research Council (http://erc.europa.eu/) since 2008.