Comparing Anytime Learning to Organic Computing

Thomas Dangl
University of Passau
Passau, Germany
dangl06@gw.uni-passau.de

***Abstract*—In environments where finding the best solution to a given problem is computationally infeasible or undesirable due to other restrictions, the approach of anytime learning has become the de facto standard. Anytime learning allows intelligent systems to adapt and remain operational in a constantly changing environment. Based on observation of the environment, the underlying simulation model is changed to fit the task and the learning process begins anew. This process is expected to never terminate, therefore continually improving the set of available strategies. Optimal management of uncertainty in tasks, which require a solution in real time, can be achieved by assuming faulty yet improving output. Properties of such a system are not unlike those present in organic systems. This article aims to give an introduction to anytime learning in general as well as to show the similarities to organic computing in regards to the methods and strategies used in both domains.**

***Index Terms*— anytime learning, autonomic computing, organic computing, artificial intelligence, intelligent systems**

I. Introduction

Anytime algorithms and by extension anytime learning are most applicable in situations that are both subject to continuous change, therefore cannot be addressed by static design and are highly time-critical. This consists but is not limited to robotics, scheduling, database query evaluation, path finding and navigation. These algorithms allow to be interrupted and provide an imperfect solution, which improves in quality as computation time progresses, thus allowing intelligent systems to gain promises about computational time in exchange for imperfect results [11].

Anytime learning refers to the approach of implementing continuous learning as an anytime algorithm in changing environments. This means that the system continues testing its strategies against the simulation model even when no new input is available. Typically it is used in the context of robotics to provide the agent with a sufficient knowledge base to operate in highly varying situations [5]. Different implementations and iterations of this technique have been developed and tested for this specific domain, which all have the automation of knowledge acquisition in common. What has yet to be explored is whether these variants of anytime learning can be applied in domains that currently are exclusive to anytime algorithms or still rely on a static computation.

Nowadays systems of high complexity across all domains require a certain degree of protection against both internal as well as external interference to remain operational. As those traits are highly similar to those of biological systems, the logical conclusion must be to build technical systems with properties of those seen in nature. This led to the somewhat recent development of Organic Computing in which design decisions are moved to run-time. In order to fulfill those promises, systems of that quality must heavily incorporate machine learning techniques [10].

The article will provide an introduction to anytime learning, the state-of-the-art strategies employed in this domain and compare the behavior and properties of this system with those of an organic system.

II. Concept and architecture

This section will deal with the general approach of anytime learning as described by Grefenstette and Ramsey [5] and the two most prominent specializations of that principle: Punctuated and case-based anytime learning. For the purpose of simplification the terminology introduced by them is used even if it is domain-related. As such the term population will entail a meaning similar to biology while agent refers to an entity in that population.

A. General approach

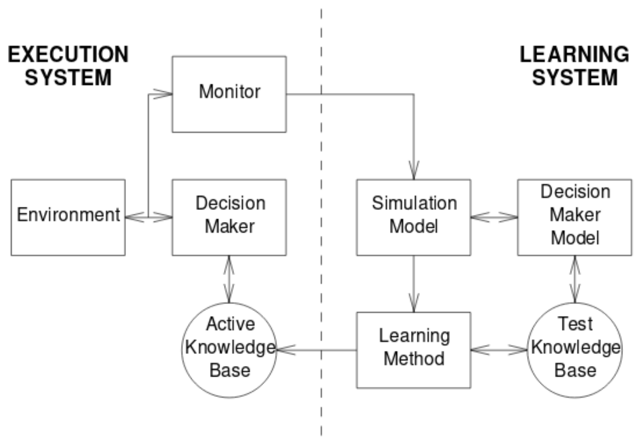

Fig. 1. Anytime learning architecture as seen in [5]

Every agent that operates on the basis of anytime learning is structured into two encapsulated modules: The execution system and the learning system. The former is responsible for interaction with the environment and other agents, it will base its actions upon the currently active knowledge base and monitors the environment to either provide new input data to the learning system or to resolve conflicts in the simulation model. The learning system will run continuously and test new strategies in the simulation model. When a strategy outperforms the currently active strategy, the learning system will update the knowledge base of the execution system, therefore providing itself new input data by testing the strategy in the environment. A generic architecture for this approach can be found in Fig. 1.

According to [5] there must be an agreement upon two policies when designing a system that uses anytime learning: When and how the learning system is reinitialized. To begin with, it must be clear under which conditions the learning system is restarted, this decision should be based upon performance metrics and parameter observation. Re-initialization is required in order to adapt the running system for a changing environment, most often this happens under the assumption that the new environment is similar to the old one. Based on the learning method in place the re-initialization might require a hard reset, which can be circumvented by using a genetic algorithm. Population after successful reset differs from implementation to implementation, a possible approach is shown in Fig. 2.

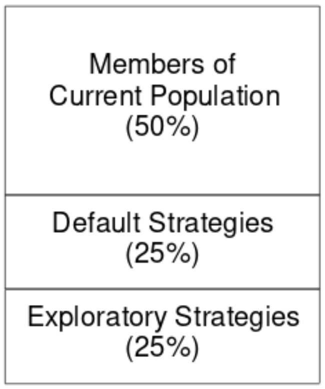
Members of Current Population (50%)
Default Strategies (25%)
Exploratory Strategies (25%)

Fig. 2. Generic initial population as seen in [5]

B. *Case-based anytime learning*

To improve the performance of the system, the genetic algorithm can be changed in a way that the initial population—in accordance to previously mentioned reset policy—contains a set of strategies that were acquired in similar environments [8]. Whenever the execution system instructs a reset of the simulation model, the best strategies that were found in this environment are stored. Later when this situation is re-encountered, the initial knowledge base can be populated with the already learned strategies. This reduces the need to relearn good and proven strategies thus improves overall quality of the solution.

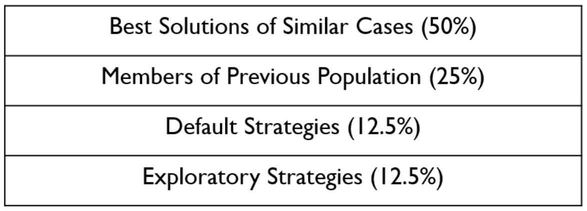
Best Solutions of Similar Cases (50%)
Members of Previous Population (25%)
Default Strategies (12.5%)
Exploratory Strategies (12.5%)

Fig. 3. Case-based initial population as suggested in [8]

Grefenstette and Ramsey introduce the use of a nearest neighbor search and weight the results of this search by how recent they are [8]. An instance of this type can be found in Fig. 3 where half of the re-initialized population consists of solutions in similar cases.

C. *Punctuated anytime learning*

In domains where the agent might get damaged or ends up incapable of acting due to its own actions, it might be favorable to physically detach the learning system from the execution system. Continuous learning still takes place in this architecture, yet there are periods of highly accelerated learning after communication with the agent that contains the execution system. The input data will be provided to the learning system either by a human operator or an automated system like a camera with image recognition. When the learning system finds a new solution, it will alert the operator to progress with the testing [7].

As a result, this reduces costs as the learning system is likely the most expensive piece of hardware. Furthermore, if the damage prevents the agent from operating, traditional anytime learning can end up in situations where one strategy consistently breaks the agent before the learning system can change its behavior. This method can even be used to co-evolve two separate populations in punctuated generations. Every individual can be matched with a highly specialized individual from another population to further improve the set of available strategies [6].

III. STRATEGIES AND PROGRESS

The previously mentioned genetic algorithm will require operators inspired by natural selection. This section will cover some of the more common genetic operators alongside conflict resolution.

A. CROSSOVER *and* MUTATION

CROSSOVER which is heavily inspired by reproduction in biology, is the most prominent binary operator for genetic algorithms. Multiple versions of this operator have been mentioned in literature, however, this article will only cover 1-POINT CROSSOVER.

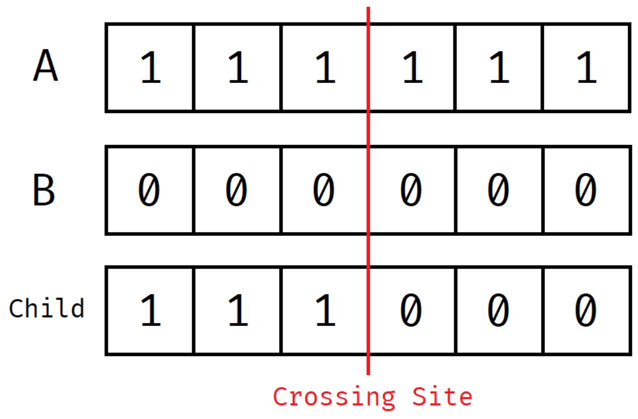

Fig. 4. 1-POINT CROSSOVER instance

As seen in Fig. 4, a random point will be selected among two parents, the child will then contain every bit on the left side of the crossover site from the first parent and every bit on the right side from the second parent.

After CROSSOVER takes place, some arbitrarily selected bits are subject to MUTATION [1]. Fig. 5 shows this in an instance where the third bit is selected and therefore flipped. Obviously both operators can be modified to work on the semantic level of the following rule model.

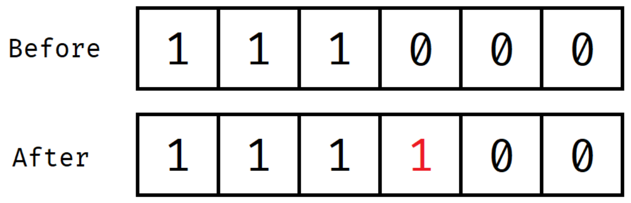

Fig. 5. MUTATION instance

B. Utility estimation

Grefenstette suggests a classifier-based rule model in which every rule is in the following form:

```
IF   the current state is in the set S
THEN apply operator O
```

To resolve conflicts, every rule in such a system has a current strength assigned to it [3]. The strength of a rule in this model is given by a function that increases with the expectation and decreases with the variance, so that it represents both confidence and utility in the current environment [4]. If the expected payoff is interpreted as a random variable, the following formula can be considered an implementation of this principle:

$$\text{Strength}(X) = \text{E}(X) - \sigma(X)$$

Given a *bid bias*, arising conflicts can now be resolved in 4 steps as shown by Grefenstette [4]:

1) Find a set of rules that nearly match the current state.
2) Define the *bid* of an action as the maximum of all strengths of the rules in the match set that specify this action.
3) Raise the *bid* to the power of the *bid bias*.
4) The *bids* form a probability distribution from which a suitable action can be picked.

Rules of low strength will be slowly removed from the population [5] as suitable replacements are introduced by the usage of the next two operators.

C. SPECIALIZE *and* GENERALIZE

The following two operators due to their nature should only be applied during high-payoff periods.

SPECIALIZE is applied when a general rule—based on a given threshold—fires. The rule's condition will be modified to more closely match the current environment, e.g. conditions given in interval representation will have their bounds moved to the arithmetic mean of the previous bound and the sensor reading, and the rule's action value is updated.

GENERALIZE is applied when a rule fires due to partial match, i.e. there is no rule that matches the sensor readings entirely. It modifies the rule's condition so it can execute given the current sensor readings, in the example of an interval this means setting either the lower or upper bound to the current value [4].

IV. RELATION TO ORGANIC COMPUTING

Now the concept of Organic Computing will be introduced and compared to that of anytime learning based on the qualitative characteristics of both approaches.

A. *Definition*

Unlike autonomic computing and anytime learning, Organic Computing is a concept that has only become relevant in the last decade. Organic Computing however does not mean to build technical systems out of organic material but to build systems that behave ”life-like”. It aims to replicate properties of living things such as flexibility, robustness and protection against internal and external interference [10]. Because biological organisms inherently require an immense level of both error tolerance and flexibility in order to endure evolution, it is highly desirable to incorporate these traits in autonomous systems to improve overall reliability. In its entirety this approach handles the ever-growing complexity of systems by shifting the focus from design-time challenges to run-time. As there is no definition that is generally agreed upon, for the purpose of this paper, systems qualify as organic not by how they are built but by how they behave and which properties they entail [10].

B. *Architecture*

An organic system consists of two complementary parts by design: One that is responsible for the technical operation of the system and the other that is contains the adaptation

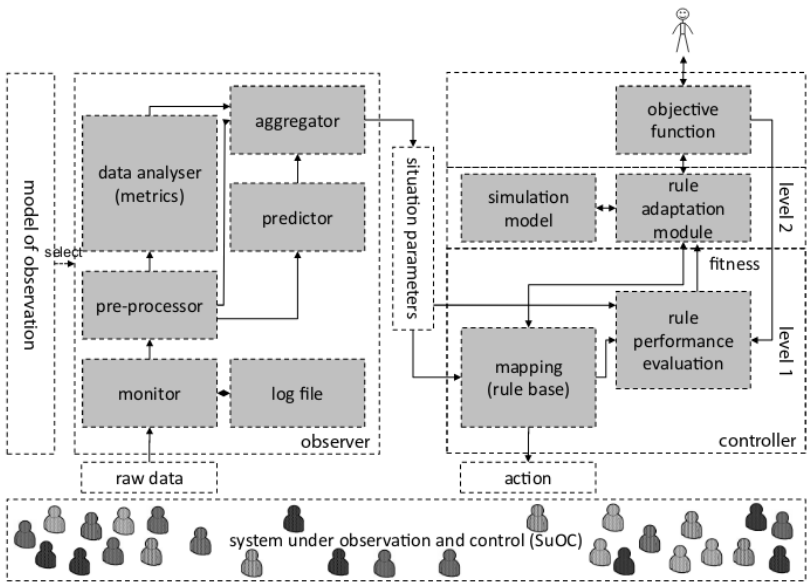

Fig. 6. Generic Observer-Controller architecture as seen in [9]

capabilities typically provided by machine learning [10]. This separation of concerns is not unlike that present in anytime learning systems, in which the learning module is responsible for knowledge acquisition and the execution module is used to interact with the environment.

The most prominent implementation of the aforementioned separation is the observer-controller architecture which can be seen in Fig. 6. It has been adopted for many scenarios such as traffic control, elevator control or cleaning robots and splits the organic system into three parts [10]:

1) The *System under Observation and Control (SuOC)* works independently of the other two components and represents the production part of the system.
2) The *Observer* monitors the actions and the state of the *SuOC* and aggregates this information.
3) The *Controller* will instruct the SuOC based on the information provided by the *Observer* and the superordinate goal of the user.

The third part—the controller—which contains the simulation model and the rule adaption model, is undoubtedly highly similar to the components of an anytime learning system [2]. It is noteworthy that the objective or utility function in this architecture is clearly user controlled, while in Grefenstette and Ramsey's generic anytime learning architecture [5] it is seen as an internal state that cannot be changed once the system is in place. In summary it can be said, that on the architectural level anytime learning and Organic Computing appear highly complementary.

C. Properties

While explaining some properties of these systems, they are to be compared to the properties of anytime learning systems. As outlined by Tomforde et al. there are nine different *self-x properties* to be found [10], this paper will only examine a few of them:

- **Self-configuration / self-adaptation:**

  Organic systems tend to configure themselves according to a superordinate goal, i.e. setting up the subsystems so itself in its entirety can achieve this certain goal. Systems that operate on the basis on anytime learning however do not change or configure their components, yet behave in accordance to a goal like organic systems. This might not be obvious at first glance, but the strength of a rule can be interpreted as the result of a utility function given the matching action. As such the utility function can be considered the higher goal for which the agent tries to optimize in the test environment. Therefore the behavior—at least partially—matches those of an organic system.

- **Self-healing:**

  In case an organic system encounters a failure—whether on the hardware or the software side—it is capable of locating and repairing it, so it eventually is able to function properly again. When the environment of an anytime learning system changes, this situation may be considered a failure as the currently present strategies no longer work and prevent the agent from operating in the new environment. It can be considered *self-healing* when the execution system reacts to this issue and re-initializes the knowledge base as explained before. This allows the agent to properly function in the new environment, repairing him so to say. Furthermore in the case of punctuated anytime learning an agent that stops operating properly due to hardware failure can be matched with a robot from a different population [6] that has the required strategies to repair the hardware defect and allow the agent to continue its task.

- **Self-protecting:**

  Organic systems must even exceed the capabilities of *self-healing* and provide protection for both the subsystems and the overall system, so that failures and attacks are detected before they take place. This allows the system to mitigate external and internal interference before any damage is caused. A certain level of protection in anytime learning systems is provided by the continuous learning and the restart mechanism of the execution system previously mentioned. When the scope of the question is extended to systems that operate under the premise of case-based anytime learning, the protection has to be considered even stronger as the initial population consists of strategies that have proven themselves to be effective in the new environment [8].

- **Self-stabilizing:**

  Eventually, an organic system must enter a stable state that is to say it must become reliable. A generic anytime learning system will build its knowledge base due to continuous learning, so over time all active rules in that knowledge base will have very similar strengths associated with them and will only be swapped out for explorative rules every now and then, which means as time progresses, the system becomes more and more confident in its environment. As such the actual progress of anytime learning can be considered inherently *self-stabilizing*.

- **Self-improving / self-optimization:**

  When speaking of organic systems a property that is often overlooked is *self-improvement*. Essentially this means that the system uses the results of previous actions to improve itself and furthermore indirectly the effectiveness of the other mechanisms. The first part of this property is self-evident for anytime learning systems as this can be considered their sole purpose. When it comes to the second part, i.e. amplifying other system properties the following is to be considered: It is clear that an improved knowledge base improves the principle mentioned when discussing *self-configuration*, so it is reasonable to assume that the learning system's capability of adapting to the environment is also improved. In regards to the *self-organization* it should be noted that the organization of the anytime learning system is not subject to change therefore cannot be improved by a wider or specialized knowledge base.

V. Summary

In this article the motivation and development of anytime learning has been briefly introduced and discussed in relation to Organic Computing. While the concept of anytime learning is much more mature than Organic Computing in its current state, it has become clear that both concepts offer a high potential for symbiosis. Looking forward the trend of ever-growing (distributed) systems in complexity and scale is only to continue. Eventually these systems will reach a state where it is no longer suitable or possible to be designed or operated by a human in the traditional sense. Therefore it is highly warranted to delegate responsibility to the run-time of such intelligent systems.

Organic Computing is suitable to fill this gap and provide a logical organization of these systems. Anytime learning seems to integrate into this concept almost intuitively. What has yet to be answered is how the adaptation of anytime learning for organic systems takes place in applications like traffic control or cleaning robots.

In conclusion it must be stated, that the concept of anytime learning will most likely be used in conjunction to existing architectures of Organic Computing in application scenarios which require an additional simulation before applying its actions in the real world.

References

[1] Nikolaos. G. Bourbakis. *Artificial Intelligence Methods and Applications*. Advanced series on artificial intelligence. World Scientific, 1992.

[2] Juergen Branke, Moez Mnif, Christian Müller-Schloer, Holger Prothmann, Urban Richter, Fabian Rochner, and Hartmut Schmeck. Organic computing – addressing complexity by controlled self-organization. *Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (isola 2006)*, pages 185–191, 2006.

[3] John J. Grefenstette. Credit assignment in rule discovery systems based on genetic algorithms. *Machine Learning*, 3:225–245, 1988.

[4] John J. Grefenstette. Lamarckian learning in multi-agent environments. In *ICGA*, 1991.

[5] John J. Grefenstette and Connie Loggia Ramsey. An approach to anytime learning. In *ML*, 1992.

[6] Gary B. Parker and H. Joseph Blumenthal. Punctuated anytime learning for evolving a team. 2002.

[7] Gary B. Parker and Karen J. Larochelle. Punctuated anytime learning for evolutionary robotics. 2000.

[8] Connie Loggia Ramsey and John J. Grefenstette. Case-based anytime learning. 1994.

[9] Sven Tomforde, Holger Prothmann, Juergen Branke, Jörg Hähner, Moez Mnif, Christian Müller-Schloer, Urban Richter, and Hartmut Schmeck. Observation and control of organic systems. In *Organic Computing*, 2011.

[10] Sven Tomforde, Bernhard Sick, and Christian Müller-Schloer. Organic computing in the spotlight. *CoRR*, abs/1701.08125, 2017.

[11] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17:73–83, 1996.