

Evolving Petri Net Situation Templates for Situation Recognition

Anders Dahlbom and Lars Niklasson
Informatics Research Centre
University of Skövde, Sweden
E-mail: {anders.dahlbom, lars.niklasson}@his.se

Abstract — Situation recognition is an important problem to address in order to enhance the capabilities of modern surveillance systems. Situation recognition is concerned with finding a priori defined situations that possibly are instantiated in the present flow of information. It can be a rather tricky task to manually define templates for situations that evolve over time, and to at the same time achieve good results with respect to recall and precision on a situation recognition task. In this paper we present some initial results concerning the task of applying genetic algorithms to evolve Petri net based situation templates of interesting situations. Our results show that it is possible to evolve Petri nets that are on par with manually defined templates. However, more research is needed in order to establish the actual effects it has on recall and precision.

Index Terms — Genetic algorithms, petri nets, situation assessment, situation recognition.

I. INTRODUCTION

SITUATION recognition is an important problem within many domains, and it is concerned with identifying a priori defined patterns of behavior in the current flow of information. The situation recognition problem is closely related to the information fusion domain, which aims at aiding decision makers in achieving enhanced situation awareness. Information fusion is generally described in terms of the Joint Directors of Laboratories (JDL) model of information fusion [1] [2]. In this model, the fusion process is basically divided into five levels of abstraction: (0) sub-object assessment, (1) object assessment, (2) situation assessment, (3) impact assessment, and (4) process refinement [1] [2].

In reference to the JDL model, situation recognition is mostly related to the situation assessment level, where the goal is to infer relational information between numerous detected objects and the environment, in the universe of interest. Lambert [3] argues that one of the main differences when moving from object to situation assessment is that we move from the numerical domain of measurable properties, into the symbolic domain where we operate on sets of facts.

In our previous work [4] [5], we have viewed situation recognition as a search for specific patterns of behavior in a state space consisting of all defined relations between all objects, over all points in time. This effectively forms our domain of interest, which consists of facts of relations over objects. It is often hard to know exactly what we are looking

for in this state space, and thus we have previously argued for the use of templates for representing the most essential aspects of an interesting situation (essential given a certain goal of the system). A template can be defined as $T = (X, C)$, where X is a set of variables for objects, and where C is a set of constraints, such as for example that certain relations between objects in X must hold, or constraints that needs to be true before other constraints. A template can be used to search the domain for all occurrences where the set of constraints hold.

An alternative to performing an exhaustive search through a state space is to use some form of state transition technique and feed the data through representations that model the behavior we are interested in. In previous work [5] we have suggested a novel approach for using Petri nets to represent situations and for managing the hypothesis space of possible matches in the situation recognition task. This approach is based on previous work by [6] [7] [8] [9]. In previous work [5] we were able to successfully recognize most of the situations we were interested in; however, the precision of the system was not very good and we thus had a high false alarm rate. In the present paper we continue our work on situation recognition, and investigate if we can use genetic algorithms for improving the precision on the situation recognition task.

The rest of this paper is organized as follows. Section two covers background work including our previously presented Petri net based approach situation recognition, as well as the basics of genetic algorithms. In section three we present an approach for using genetic algorithms to evolve Petri nets. Section four contains a few initial experiments and section five discusses the results and suggests improvements. Section six concludes the paper and points out our road ahead.

II. BACKGROUND

A. Petri nets

A Petri net is according to Murata [10] a directed, weighted, bipartite graph in which nodes come in two flavors: places and transitions. Edges in the graph either connect places to transitions (called input arcs), or transitions to places (called output arcs). The final component of Petri nets, tokens, are used to denote the existence of a process (or similar) at specific places in the net. Places are thus containers of tokens, and the global state, which is called the marking, consists of all tokens within the net. Transitions in a Petri net changes the marking by consuming tokens at input places and producing

tokens at output places. In order for a transition to be activated, each input place must contain a specified number of tokens. In a graphical notation, places are drawn as circles, transitions as vertical bars, and tokens as dots inside places.

Petri nets are according to Sowa [11] a generalization of finite state automata (FSA) and can be viewed as a combination of FSA and event charts, where places correspond to states of FSA, and where transitions corresponds to events in a flow charts. Further, Sowa [11] argues that the main strength of Petri nets is their ability of representing parallel and concurrent processes.

B. Petri nets for recognition

Petri nets are not usually thought of as mechanisms for recognition (although their relative finite state machines (FSMs) are). Castel et al. [8] argue that Petri nets actually are quite suited for this task since they allow for sequencing, parallelism, and synchronization to be easily represented and visualized. This can be of major importance when a decision maker is to understand the underlying components of a recognized situation, as well as when defining what we are interested in recognizing through the use of expert knowledge.

A reason for using Petri nets instead of FSMs and their probabilistic counterpart hidden markov models (HMMs), is their ability of representing parallel flows of information. HMMs and FSMs operate on strictly sequential data (with the exception of hierarchically decomposed models).

In our previous work [5], we have proposed a novel approach based on Petri nets, for recognizing situations and for managing the space of partial matches coupled to situation recognition. This approach extends previous work by [6] [7] [8] [9], and will briefly be presented in this section.

Representation

Two types of transitions are used in the approach. The first is the basic Petri net transition, for which the requirements for the transition to be activated is that a specified number of tokens exists in each of its input places. The second type of transition is a conditional transition, introduced by Ghanem et al [6], to which a constraint can be assigned. As an example, the constraint $\text{approach}(x, y) = \text{true}$ can be assigned to a transition. This transition would be activated in case an event of that type and value is processed in the system and if the correct number of tokens is available at the input places. Every conditional transition maps to exactly one constraint in a template for an interesting situation.

Tokens in the approach are used to represent partial matches between a template and the stream of events that we have observed. This follows the ideas of coloured Petri nets, in which tokens are used as carriers of information. Recall, a template $T = (X, C)$ consists of a set of variables X and a set of constraints C . Likewise, a token represents parts of a template $TO_o = (X, C')$, where C' is a subset of C which consist of all non temporal constraints. The temporal constraints do not need to be modeled in a token since the structure of a Petri net contains the temporal constraints in a template.

Places represent partial stages of the matching procedure with a specific situation type, and a token at a specific place means that there is a partial match that has reached that far in

the matching procedure. In order to keep the complete matching space available, transitions do, in contrast to conventional Petri nets, not consume tokens when they are activated. An example of a Petri net describing an imaginary pick-pocket situation, is given in figure 1.

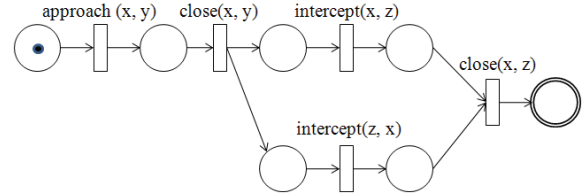


Fig. 1. Illustration of a petri net for recognizing a “pick-pocket” situation. This specific petri net has previously been used in [5]. In order for a pick-pocket situation to be recognized, object x must first approach object y , after which objects x and y are close, after which objects x and z intercept each other, after which they are close to each other.

Algorithm

Each event that is processed by the Petri net needs to be tested against each conditional transition. In case the event matches the constraint and value for a transition, a new token representing a partial match is created, in which the constraint for the transition is bound, and in which the variables denoted by the constraint are bound. The next step consists of trying to combine the new token with each combination of tokens available at the input places. A valid combination consist of one token from each input place, for which already bound variables and constraints in any of the tokens must not stand in conflict with each other (for example, one token having x bound to 3 and another token having x bound to 5, cannot be combined since there is a conflict). The combination of a set of tokens is formed by taking the union of all variable and constraint bindings.

After having processed a new event in all transitions (and formed a list of all valid combinations at each transition), the newly formed combinations are propagated through the network through the output places of each transition (propagation here allowing for missed detections).

Since tokens are not consumed by transitions in the suggested approach, the space of partial matches grows rather quickly. To counter for this, we have suggested using a sliding window approach to remove tokens that have not been updated within a certain period of time.

The initial marking of the Petri net is constructed by inserting an empty token (no variables and constraints are bound) at each place which does not have any incoming edges to them. This allows us to implicitly manage role assignment in the Petri nets. Possible instantiations of the modeled situation in the data stream can be found in specific matching places, which do not have any outgoing edges. These are also denoted with double circles in graphical notation.

Example

To illustrate the usage of Petri nets for situation recognition, we will shortly go through an example. This is illustrated in figure 2, in which changes in the marking of a Petri net is illustrated as 4 different events are processed. The initial marking of the net consist of a single token in the input place (figure 2a).

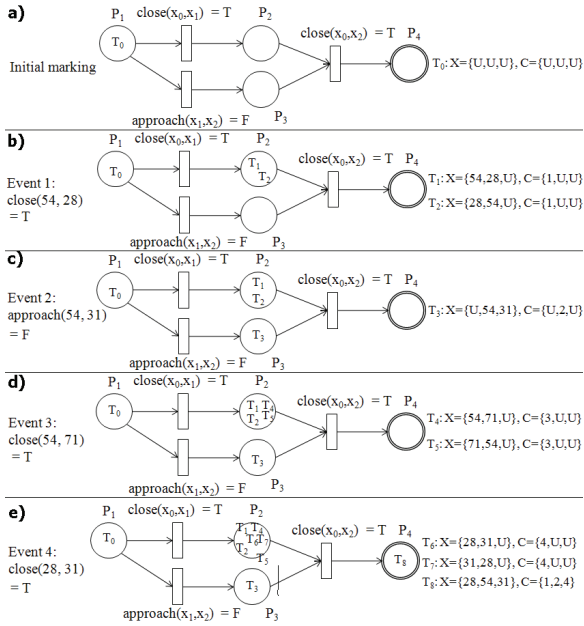


Fig. 1. Illustration of the change of the marking of a Petri net when processing events in a simple Petri net.

When the first event, $\text{close}(54, 28) = \text{true}$, is processed it matches the first and third transitions. In the first transition, a new token is created, in which $x_1 \leftarrow 54$, $x_2 \leftarrow 28$, and $c_1 \leftarrow 1$. The new token is combined with the empty token in the input place, and the result is a new token in place 2. Close represents a symmetrical relation, and in order to allow for all possible matches, a second token is created in which $x_1 \leftarrow 28$, $x_2 \leftarrow 54$, and $c_1 \leftarrow 1$. This token is also put in place 2. Similarly, two new tokens are created in the third transition, but now, $x_2 \leftarrow 54$, $x_3 \leftarrow 28$, and $c_3 \leftarrow 1$, and vice versa. However, since there are no tokens available in places 2 and 3, the new tokens cannot be combined and are removed. The marking after the first event is illustrated in figure 2b.

Upon processing event 2, $\text{approach}(54, 31) = \text{false}$, a new token is formed in the second transition, and it is then combined with the empty token. This results in a new token in the third place. In this token, $x_1 \leftarrow 54$, $x_2 \leftarrow 31$, and $c_2 \leftarrow 2$. The marking after processing the event is shown in figure 2c.

The third event up for processing is $\text{close}(54, 71) = \text{true}$. This time, two new tokens are created in the first transition (similar to before). When the event is processed by the third transition, two new tokens are created in which $x_2 \leftarrow 54$, $x_3 \leftarrow 71$, and $c_3 \leftarrow 3$, and $x_2 \leftarrow 71$, $x_3 \leftarrow 54$, and $c_3 \leftarrow 3$. This time, there are tokens available in both places 2 and 3. If we look at the token in place 3, x_1 is unbound, $x_2 = 54$, and $x_3 = 31$. The two new tokens cannot be combined with this token since $x_3 = 31$ and not 54 or 71 (as in the new tokens), and hence, they are removed due to a conflict in variable bindings. The marking of the net after event 3 is illustrated in figure 2d.

Finally, we have the fourth event, $\text{close}(28, 31) = \text{true}$. This event also results in two new tokens in place 2, however, this time one of the two new tokens created in the third transition does not stand in conflict with a token in place 2 and a token in place 3. Therefore, a combined token is placed in place 4. In this token, $x_1 \leftarrow 28$, $x_2 \leftarrow 54$, $x_3 \leftarrow 31$, $c_1 \leftarrow 1$, $c_2 \leftarrow 2$, and $c_3 \leftarrow 4$. Hence, a match has been found. The resulting marking of the net is illustrated in figure 2e.

Problem

In our previous work [5] we have shown that it is possible to find interesting behaviors in a simulated pick-pocket scenario, however, the precision was not very good. A reason for this can be that the template that was used to search for the behavior was handcrafted. The task of manually constructing templates for complex behavior is, naturally, a complex task. The designer needs to have very precise knowledge of what the modeled behavior consists of, and what it does not consist of. Furthermore, the world is a highly dynamic place and behaviors change, either naturally or by the will of the entities exerting the behavior. This is especially true in surveillance applications, as the behaviors we often are interested in finding are exerted by people or groups that consciously try to avoid detection.

In this paper we carry out some initial investigations regarding the possibility of improving the precision of Petri nets for situation recognition through the use of genetic algorithms. Genetic algorithms have in the past couple of decades successfully been applied in many domains to search for complex solutions in large search spaces. Our research question in the present paper thus reads: *is it possible to improve the performance of the Petri net based situation recognition task through the use of genetic algorithms.*

C. Genetic algorithms

Genetic algorithms are inspired by the Darwinian principles of evolution and survival of the fittest, and according to Luger [12]; learning is viewed “*as a competition among a population of evolving candidate problem solutions*”. In light of machine learning, a genetic algorithm is simply a methodology for finding a good hypothesis h in a hypothesis space H , which is inspired by Mother Nature. One advantage of genetic algorithms is however that they search at many places of the hypothesis space simultaneously.

A population of candidate solutions is formed, where each is encoded as a set of genes that together form a genome (or chromosome). Each gene is often encoded as a single bit, but numerical values are also popular. The population evolves over a number of generations, where in each generation a number of genetic operators are applied to the population in order to evolve the next generation of individuals. More specifically, for each generation, each candidate solution in the population is evaluated on the problem at hand, in order to establish a fitness value that describes how good it is at solving the problem. After this, a selection and reproduction phase starts, in which a number of individuals in the present population are selected, based on their fitness, to form the basis for the next generation of candidate solutions.

In the selection and reproduction phase of a genetic algorithm, there are many different schemes that can be used, or which can be used in combination with each other. To name a few, genetic cross over combines two genomes to construct a new, mutation flips some of the genes, roulette-wheel selection selects two individuals for reproduction based on their contribution to the summed fitness of the population, and elitism selection clones the best performing individuals in the present population. For more information regarding genetic operators, selection, and reproduction, c.f. [12] [13].

III. IMPROVING PRECISION WITH GENETIC ALGORITHMS

In this section we will discuss the approach of applying genetic algorithms to evolving Petri net situation templates that we have investigated in this paper.

A. Symbolic mapping

In order to allow for easily applying the genetic operators, it is suitable to have a genetic representation in binary or numeric form. This allows us to easily implement a cross over at any specific point in the genetic sequence, as well as allowing for mutation of single bits or genes. We cannot easily evolve the previously described Petri nets for situation recognition in their present representation, and we therefore need to map the Petri net representation into a genome representation consisting of bit strings.

In our initial experiments we only consider a genetic representation of static size, i.e. the length of the genome does not change. In order to achieve this, we must decide on the maximum number of places, transitions, and free variables for objects that we can model. What is left for the evolutionary process to operate on, is the types of events that transitions represent, the truth value they represent, the variable bindings in each transition, which places are considered match and input places, respectively, and finally, which input and output edges that are active. Figure 3 illustrates the suggested bit string representation of a Petri net for situation recognition.

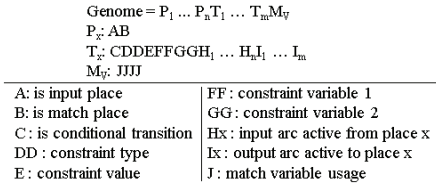


Fig. 3. Bit string representation of a Petri net for situation recognition, which consist of n places, m transitions, and a maximum of 4 variables.

In order to properly understand the bit string representation, figure 4 illustrates an example where we have mapped a simple Petri net to a bit string.

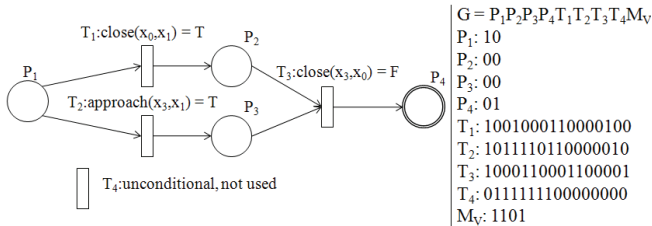


Fig. 4. An example of how to convert a Petri net to our suggested bit string representation. To the left we see a simple Petri net, and to the right its genome representation in binary format.

As can be seen in the figure, place 1 is marked as input place with bits 10, place 2 and 3 are neither input nor match places, thus 00, place 4 is marked with 01, since it is a match place. Transitions 1-3 use conditions, thus the first bit of 1. Transitions 1 and 3 have the close relation, which is represented with 00, and transition 2 has an approach relation represented with 01. Transition 1 has variables x_0 and x_3 , thus 00 and 11, respectively, it only has place 1 as input which is represented by 1000, and it has place 3 as output place: 0010. Similar logic applies to the other transitions. Finally, we have match variable activation 1101, which means that only variables x_0 , x_1 , and x_3 are used in an interesting situation (x_2 is

thus neglected when considering matches). In the experiments, we have used a static representation with 8 places, 8 transitions, and 4 variables.

B. Genetic algorithm

The problem solving task that we would like to evolve solutions to, consist of processing events in order to recognize when certain situations occur. An algorithm would run for a specified number of generations, in which each individual in the population is fed the event data. After having parsed all data through the Petri nets that each individual represent, a fitness value is calculated as follows:

$$f = 0.5R + 0.5P, \quad (1)$$

where R is the recall and P is the precision, of the solution that the individual represents. Recall and precision are defined as follows:

$$R = TP / (TP + FN), \quad (2)$$

$$P = TP / (TP + FP), \quad (3)$$

where TP is true positives, FN is false negatives, and FP is false positives of the recognition problem.

When a fitness value has been calculated for all individuals, a new generation of problem solutions is evolved by using a combination of elitism selection, roulette-wheel selection, genetic cross-over, and mutation.

C. Bootstrapping

Although one might argue that evolution in nature started from a single celled organism and that we should copy this procedure to its full extent, we believe that the inclusion of prior knowledge possibly can enhance the evolutionary process. Evolutionary algorithms are in themselves often used as a bootstrapping method for other algorithms. We however intend to bootstrap the evolutionary process itself. In the classical case, the genomes of all individuals in the first population are randomized. When bootstrapping the process however, we would like to make use of already existing knowledge and information to create an initial population that already is, to some extent, good at performing the problem solving task that we are interested in solving.

In the present paper, we suggest to bootstrap the evolutionary process with manually created Petri net representations. However, to also introduce some variation into the discovery process, we also suggest mutating each gene in the genome of each bootstrapped individual, according to some pre defined probability.

IV. EXPERIMENTAL RESULTS

The suggested approach for applying genetic algorithms to evolve Petri nets that describe interesting behavior has been implemented in our recognition platform previously presented in [5]. The procedure has been tested on a simulated pick-pocket scenario, which has been developed in a simulator previously presented in [14].

A. Simulation setup

The scenario consists of a number of pedestrians that move around in a shopping zone, where a number of stores attract

pedestrians to move towards them. This results in a rather noisy environment, in which to do the recognition. On top of this normal model, pick-pocket situations are instantiated by allocating two thieves that move through the environment. After a random amount of time, one of the thieves selects a suitable pedestrian as a target, and starts moving towards this target to intercept it. A successful intercept is interpreted as the thief successfully having picked the pocket of its target, whereby the thief and his/hers accomplice moves towards each other to hand the stolen goods over. A successful handover completes the pick-pocket situation. This behavior is repeated at random points in time throughout the scenario. A more detailed description of the scenario is available in previous papers [4] [5] [14].

The output from the simulator consists of tracks, of all objects that move around in the environment, sampled at a frequency of 4 Hz. The track data (perfect tracks at this point in our research) is analyzed to extract three different types of relations that can exist between the different objects: $close(x, y)$ – two objects being close to each other, $approach(x, y)$ – object x moving towards object y , and $intercept(x, y)$ – object x on an intercept path with object y . These relations are described in more detail in previous work [5] [4]. When a relation between a pair of objects is found (which has not already been found), an event with this information is passed on to the recognition system. Likewise, when a relation that previously was true no longer holds, an event is also passed on to the recognition system.

The simulator has been used to construct 60 data sets, where each is formed from a 10 minute long scenario. The first half (1-30) of these data sets has been used for evolving Petri nets, and the second half (31-60) has been used for evaluation purposes. Two different experiments have been carried out, where the first involves evolving Petri nets from a randomly seeded population of solutions, and where the second involves bootstrapping the initial population of solutions with a manually defined situation template.

B. Evolution from scratch

In this experiment, the genetic algorithm has been run for 300 generations with a population of 100 individuals. A fitness for each individual has been calculated using equation 1, after which two different selection/reproduction schemes have been used: 10% of the new population is selected with elitism, and 90% of the new population has been created by selecting two individuals with roulette wheel selection, and then these have been combined with a random single-point crossover. After the selection procedure, each gene in each genome of the new individuals, goes through a mutation procedure, where each gene has a $1/(\text{length of genome})$ probability of being mutated. Having a probability of $1/(\text{length of genome})$ has been found successful in many situations [13]. Individuals selected with elitism selection are excluded from the mutation procedure. It can often be important to have some elitism in the selection procedure, in order to ensure that high ranked solutions are not lost along the way. In the experiments, we have used four randomly selected data sets to evaluate each generation of individuals.

The results of the evolutionary process are presented in

figure 5, which contains the best fitness, precision and recall, when evolving a Petri net from a randomly seeded population of individuals. As can be seen in the figure, the best performing network reaches fitness between 0.5 and 0.6, recall of around 1, and a precision slightly below 0.1.

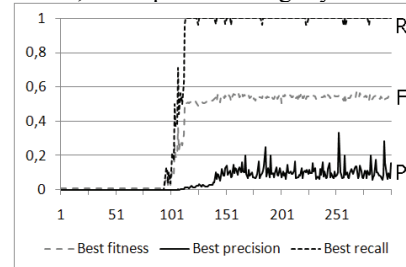


Fig. 5. Results obtained when evolving with a randomly seeded population.

C. Bootstrapping

In this experiment, the genetic algorithm has also been run for 300 generations with a population of 100 individuals. The same selection and mutation scheme as in the first experiment has been used. However, this time the initial population has been seeded with the manually defined Petri net as follows. For each individual, the manually created Petri net is converted to a genome, which undergoes a mutation procedure, where each gene has a 0.01 probability of being mutated. The reason for mutating the genomes randomly is to induce some variation into the initial population, and the choice of using a higher probability of mutation compared to that used in the evolutionary process, is to ensure that there actually is some diversity in the population. The results of the evolutionary process with bootstrapping are presented in figure 6. Average fitness, average recall, and average precision do not change to any significant extent; however, there is a slight increase when looking at the trend from generation 0 to 300 for each of the three included measures.

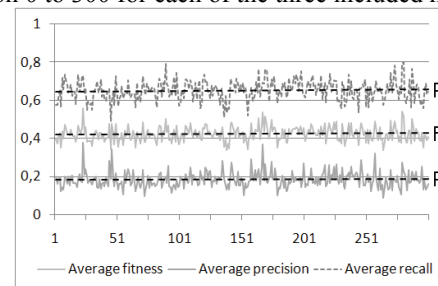


Fig. 6. Results obtained when evolving with bootstrapping.

D. Comparison

The best evolved networks from both experiments have been evaluated on the remaining 30 data sets, and are compared with each other and with the manually designed template in table 1, in which TP, FP, and FN denotes true positives, false positives, and false negatives, respectively. As can be seen in the table, the evolved network has higher recall than the manually designed network and the network evolved with bootstrapping. The precision is however lower. The network that has been evolved with bootstrapping has a slightly lower recall and a slightly higher precision, compared with the manually designed network. The differences between the networks are however not of significant extent.

TABLE I
RESULTS WHEN APPLYING THREE DIFFERENT NETWORKS FOR RECOGNITION

		TP+FP	TP+FN	TP	Recall	Precision
Handcrafted	Mean	18.63	7.03	6.00	0.86	0.35
	Std dev	5.92	1.07	1.45	0.12	0.13
Evolved	Mean	83.00	7.03	6.97	0.99	0.09
	Std dev	28.48	1.07	1.07	0.03	0.04
Bootstrapped	Mean	15.8	7.03	5.87	0.84	0.41
	Std dev	5.50	1.07	1.22	0.13	0.15

V. DISCUSSION

In the experiments we have shown that it is possible to evolve Petri nets for situation recognition. However, the precision of the evolved networks is not higher than that of our manually designed network. We have thus not been able to answer our research question properly, since we cannot say that it is not possible to increase the precision through the use of genetic algorithms. Neither can we say that it is possible to increase the precision to a significant extent.

Currently we are mainly focusing on defining what the behavior we are trying to find consists of, naturally. However, it can be of equal importance to consider what the behavior we are looking for does not consist of. In its current specification, the Petri nets that we use do not have the capability of representing what does not constitute a match. We therefore propose to include this, and in addition to match places, we would have something that we term not-match places. Hypotheses ending up in a not-place, indicates that it is not a match, and can therefore be removed.

Furthermore, the complete set of potential hypotheses for matching is kept (with exception of those passing beyond the sliding window). This also poses a problem, as even though we would include not-match places, the hypotheses they are build from still remain active in the network. In its current specification, all transitions put all tokens they consume back to the places where they came from. Instead, we would like to handle this dynamically, so that the evolutionary process determines which transitions that consume their input tokens, and which that do not. In this way, the evolutionary process would also evolve how the hypothesis space of candidate matches is managed, in addition to the Petri net structure.

Finally, the fitness function in the present study promotes solutions equally based on their recall and precision. However, the aim of this paper was to investigate if we could improve on the precision of the situation recognition problem. This indicates that we perhaps should have used a fitness function that promotes precision over recall, e.g. $f = 0.75P + 0.25R$. The results when using such a fitness function would most likely turn out different, and will thus be elaborated on in our future work.

VI. CONCLUSION

In this paper we have shown that it is possible to use genetic algorithms to evolve Petri nets for situation recognition. We have however not been able to show that the precision of the recognition task can be improved. This remains an open question which we aim at pursuing.

Even though it is a complex task to evolve Petri nets for situation recognition, we still believe that genetic algorithms

can play an important role when constructing templates of interesting behavior.

In future work we will continue our investigation of how to improve the precision of situation recognition algorithms. This is very important since the world is constantly changing and since it is very hard to manually construct templates for exactly what we are interested in finding. In addition to investigating genetic algorithms, we will investigate the potential of genetic programming and supervised symbolic learning. Our first step however concerns investigating the changes we have proposed in this paper.

ACKNOWLEDGMENT

This work was supported by the Information Fusion Research Program (University of Skövde, Sweden) in partnership with Saab Microwave Systems and the Swedish Knowledge Foundation under grant 2003/0104.

REFERENCES

- [1] A. Steinberg, C. Bowman, and F. White, "Revisions to the JDL data fusion model", in *Sensor Fusion: Architectures, Algorithms, and Applications*, proceedings of the SPIE, Vol. 3719, Orlando, FL, USA, 1999.
- [2] J. Llinas, C. Bowman, G. Rogova, A. Steinberg, E. Waltz, and F. White, "Revisiting the JDL data fusion model II", in proceedings of the 7th International Conference on Information Fusion, Stockholm, Sweden, 2004.
- [3] D. Lambert, "An exegesis of data fusion", in Reznik, L and Kreinovich, V. (eds.), *Soft Computing in Computing and Information Acquisition*, Springer Heidelberg, 2003.
- [4] A. Dahlbom, L. Niklasson, G. Falkman, and A. Loutfi, "Towards template-based situation recognition", in proceedings of the SPIE Symposium Defense, Security, and Sensing, Vol. 7352, Orlando, FL; USA, 2009.
- [5] A. Dahlbom, L. Niklasson, and G. Falkman, "Situation recognition and hypothesis management using petri nets", accepted for presentation at MDAI 2009, to appear in LNAI Vol. 5861, pp. 303-314, to appear.
- [6] N. Ghanem, D. DeMenthon, D. Doermann, and L. Davis, "Representation and recognition of events in surveillance video using petri nets", in proceedings of the Conference on Computer Vision and Pattern Recognition Workshop (CVPRW '04), pp. 112-120, 2004.
- [7] G. Lavee, A. Borzin, E. Rivlin, and M. Rudzsky, "Building petri nets from video event ontologies", in proceedings of the third International Symposium Advances in Visual Computing (ISVC '07), Springer Berlin / Heidelberg, Vol. 4841, pp. 442-451, 2007.
- [8] C. Castel, L. Chaudron, and C. Tessier, "What is going on? A high level interpretation of sequences of images", in proceedings of the 4th European conference on computer vision, workshop on conceptual descriptions from images, Cambridge, UK, 1996.
- [9] M. Perše, M. Kristan, J. Perš, and S. Kovačič, "Recognition of multi-agent activities with petri nets", in proceedings of the 17th International Electrotechnical and Computer Science Conference, Portorož, Slovenia, 2008.
- [10] T. Murata, "Petri nets: properties, analysis and applications", in *Proceedings of the IEEE* 77(4), pp. 541-580, 1989.
- [11] J. Sowa, "Knowledge representation: logical philosophical, and computational foundations", Brooks/Cole Thomson Learning, Pacific Grove, 2000.
- [12] G. F. Luger, "Artificial intelligence : structures and strategies for complex problem solving - 4th edition", chapter 11, Pearson Education Limited, Essex, England, 2002.
- [13] B. Schwab, "AI game engine programming", chapter 20, Charles River Media, Inc., Hingham, Massachusetts, 2004.
- [14] A. Dahlbom, L. Niklasson, and G. Falkman, "A component-based simulator for supporting research on situation recognition", in proceedings of the SPIE Symposium Defense, Security, and Sensing, Vol. 7352, Orlando, FL, USA; 2009.