Graph Representations for Evolutionary Algorithms

Holger Mauch

March 4, 2002

Abstract

Many different representations have been suggested as the underlying data structures, upon which the genetic operators act. Among the most prominent examples are the evolution of binary strings, real-valued vectors, permutations, finite automata, and parse trees. In this project the use of graphs as the data structures that undergo evolution shall be examined. Once suitable mutation and recombination operators have been designed for graphs, then these concepts can be transferred to evolve Petri Nets, which structurally can be considered as specialized bipartite graphs. The ultimate goal would be to breed Petri Nets, which in their extended version (adding inhibitor arcs) are as powerful as Turing machines. Evolving Petri Nets could then be compared to Genetic Programming (GP), where computer programs are evolved.
Contents

1 Project Overview 4
   1.1 Topic of the research project .......................... 4
   1.2 Project Plan for 8 weeks ............................. 4
   1.3 Desired Results and Future Research Plans .............. 5

2 Existing Literature and Research in this Area 6
   2.1 Literature involving EC and PN .......................... 6
   2.2 Literature involving EC and Graphs .................... 6

3 Optimization Problems with Graphs as Solutions 6
   3.1 RealTSP: a more Realistic TSP .......................... 6
   3.2 Communication Network .................................. 7
   3.3 Chemical Structures ...................................... 7

4 Ideas for Evolving Graphs 8
   4.1 Graph Definition used here ............................. 8
   4.2 Data Structures for Graphs .............................. 8
      4.2.1 Adjacency Matrix .................................... 8
      4.2.2 Adjacency List ...................................... 9
      4.2.3 Incidence Matrix .................................... 9
      4.2.4 Direct Representation with Explicit pointers ...... 9
   4.3 Genetic Operators for Graphs ............................ 9
      4.3.1 Mutation for Graphs ................................. 9
      4.3.2 Recombination for Graphs .......................... 10

5 Interpretation of an optimal solution 11

6 Petri Net Overview 11
   6.1 The Origin of Petri Nets ................................ 11
   6.2 Place/Transition Nets .................................... 12
   6.3 Application areas of Petri nets ......................... 14
   6.4 Summary of Petri net properties ....................... 14

7 The Idea of Evolving Petri Nets 15
   7.1 Genetic Operators for Petri Nets ....................... 16
      7.1.1 Mutation for Petri Nets ............................ 17
      7.1.2 Recombination for Petri Nets ....................... 17

References 18
List of Figures

1 Place/Transition Nets Summary ..................... 13
1 Project Overview

For this project, first I had planned to investigate the use of Petri nets (PN for short, see section 6) as the data structure that undergoes evolution. To be more specific, I had envisioned to use GA (rather than ES, or EP) to breed Petri nets - in a similar fashion to how GP uses GA to evolve computer programs. After starting the literature research however, it has become more and more clear to me that there are almost no publications that deal with this kind of approach. There exist very few papers that deal with both evolutionary computation and Petri nets. Even if a paper deals with both EC and PN concepts, such as [9], then most often EC plays only a minor role in the optimization of a certain special feature of the PN. In [9], for example, the GA merely influences the firing rules, but not the structure of the PN. After browsing the abstracts of several of these papers, my impression is that some of the EC and PN concepts have been employed together, but none of these papers would follow the rigorous way of using PNs as the data structured to be evolved.

1.1 Topic of the research project

Since there are no published results upon which I could base the research I had originally envisioned, I decided to start several steps further back. What needs to be examined first is, how can graphs in general be used as the data representation that undergoes evolution. Syntactically, PNs are bipartite graphs. The main interest in this project is therefore to describe how GA can be made to work on individuals, which are graphs. When I read in chapters 11 and 19 of [1] about the recombination of tree structures in GP, it appeared to me that there are many interesting ways of generalizing the recombination to arbitrary graphs, not just trees. Since I imagine that recombination is an important genetic operator when it comes to evolving graphs, my approach follows rather the GA stream within EC, rather than ES, or EP. In particular, I suspect that for most problems, there is a chance that the whole solution can be decomposed into subsections (building blocks) of a problem.

1.2 Project Plan for 8 weeks

1. Browse through the literature that deals with the evolution of rather complex data structures somehow resembling graphs. A good starting point might be the references given in [1] in the sections where finite
automata are chosen as the data structures undergoing evolution.

2. Design an example optimization problem, whose solutions can be represented as graphs. This example can be used to illustrate how to evaluate the fitness of an individual (i.e. a graph), and how the proposed genetic operators act on it. See section 3 for my ideas of an example.

3. Formalize my existing ideas of how to mutate and crossover graphs (see subsubsections 4.3.1 and 4.3.2 for an outline of my ideas.)

This 8 week project should result in a conclusion that says either

- Yes, the idea of breeding Petri nets could be a promising approach. The genetic operators designed for graphs are reasonable, and it should be possible to adapt the concepts that have been developed for graphs to Petri nets as well.

- No, the idea of breeding Petri does not look promising. This is because [...] enter reasons [...]. (And this is probably why there are no publications in the literature that deal with this idea.)

1.3 Desired Results and Future Research Plans

The desired result would be a conclusion saying that the idea of evolving Petri nets is worth pursuing further. The 8 week milestone would be to have the design of a mutation and recombination operator for graphs, which can readily be extended to the structure of Petri nets. Another goal is to list, in an organized way, papers and published articles that deal with both Petri nets and evolutionary computation, such that any ideas that somehow resemble that of breeding Petri nets can be extracted and discussed for future research plans.

Further ahead in the future would then be plans to evaluate the fitness of Petri nets in a time efficient way. How can existing Petri net simulators be used for that purpose? Implement a system that actually allows the evolution of Petri nets. If the results of the evolution process are not satisfying, which parameters should be changed to improve the process? Which practical problems, apart from rather trivial academic problems, have solutions that can be expressed as Petri nets?
2 Existing Literature and Research in this Area

2.1 Literature involving EC and PN

As mentioned above, a literature search for documents containing any of the following pairs of keywords such as (EA,PN), (EC,PN), or (GA,PN) did not result in any papers directly using the idea of breeding Petri nets.

[9] are using the GA concepts merely to optimize the dynamic behavior of the PN, which represented the solution to a highly constrained scheduling problem.

I compiled a list of around 15 more references that use both the EC concept and the PN concept. After browsing through the title and/or abstract, they do not seem to breed PNs.

2.2 Literature involving EC and Graphs

At this point, I am not aware of any publications that deal with evolutionary computation using graph data structures, or similar representations as individuals.

The literature search for EC and graphs is still incomplete. I plan to check some original papers referenced in the textbook, especially those dealing with crossover of finite automaton.

3 Optimization Problems with Graphs as Solutions

Most of the well known optimization problems that have a graph as a solution do not fit my intentions of evolving graphs of various sizes well. For example, the traveling salesman problem (TSP) is solved by a graph that has a fixed number of edges, and a fixed number of vertices. I tried to design example problems, whose solutions have an arbitrary number of edges, and a fixed number of vertices. It would be desirable to find more problems that have as a solution an arbitrary number of edges and an arbitrary number of nodes. Suggestions for such optimization problems are very welcome.

3.1 RealTSP: a more Realistic TSP

The following is a variant of the well known TSP. My intention was to find a problem, whose solution does not have a predetermined number of edges, or nodes.
As in the original TSP, there is a set of \( n \) cities \( C = \{c_1, \ldots, c_n\} \). Some pairs of cities are connected with travel links. Each of the links is assigned a certain cost. (If two cities are not directly connected, assign the cost \( \infty \) to it.) This environment can be considered as a weighted graph \( G \).

In contrast to the original TSP, cities can be visited once, more than once, or not at all. Links can be travelled once, more than once, or not at all. Every time a link is travelled, it causes cost according to the edge weight. Also, each city \( c \in C \) is assigned a potential revenue \( r(c) \) according to the revenue function \( r : C \to \mathbb{R} \). A city \( c \) that is visited at least once yields revenue \( r(c) \) for the salesman. Visiting a city more than once does not result in additional revenue. At the end, the salesman wants to return to the city where he started from [still thinking whether I should include that constraint or not]. The objective of the salesman is to maximize profit, calculated as the difference between total revenue and total cost.

The individuals in the population undergoing evolution would be connected subgraphs of \( G \); links, which are traveled multiple times, will be represented as weighted edges in the solution graph. The idea is that hopefully highly profitable subgraphs will be recombined as evolution progresses. Evolutionary computation should be able to jump through local pits (unprofitable cities that lie on the way between high profitable areas).

I plan to use this problem (named RealTSP) for the illustration of genetic operators during the course of my project.

### 3.2 Communication Network

The design of a communication network, e.g., a computer network, consists of nodes (computers, switches, routers, etc.) and links (cables, radio links, satellite links, etc.). Therefore the network can be considered as a (multi)graph. The problem of designing a network that satisfies certain properties (min. cost, max. reliability, min. environmental impact, etc.) can be considered as an optimization problem.

### 3.3 Chemical Structures

Designing chemical molecules is an example that has an arbitrary number of edges (bonds) and an arbitrary number of nodes (atoms). The evolution of chemical molecules would involve the recombination of molecules that have been selected according to some fitness function that measures certain desired properties of the molecule. The value of these properties must be derived from the structural formula of the molecule. Due to the laws of
physics and chemistry I expect these kind of optimization problems to be highly constrained.

4 Ideas for Evolving Graphs

4.1 Graph Definition used here

Since I have the purpose in mind to extend the ideas developed for graphs to Petri nets (place/transition nets to be precise) later on, I want the graphs considered here to be

- directed
- weighted
- loop-free (no edges from a vertex to itself)

For example, in a solution graph to the problem RealTSP designed earlier an edge weight of $k$ for an edge going from city $c$ to city $c'$ means that the salesman's schedule involves traveling $k$ times from city $c$ to city $c'$.

The Petri net data structure intended for future research will be a directed, weighted, bipartite graph, and hence loop-free.

4.2 Data Structures for Graphs

To some extent, it matters how the graph representation used in the EA is actually represented on the implementation level ([2], chapt. 24). Some data structures implement graphs more or less efficiently on a computer. If we want an acceptable running time of our EA, it is important that the genetic operators designed in 4.3 can be performed efficiently.

4.2.1 Adjacency Matrix

The advantage of the adjacency matrix representation is that using matrices as the individuals that undergo evolution has already been examined to some extent (see [1], section 17.5).

The main disadvantage is that if we want to allow for a variable number of nodes in the individual graphs, this representation is somewhat clumsy. Dynamically increasing the size of matrices seems to cost a lot of time. Statically allocating matrices that have as the number of rows and columns the upper bound of max. nodes allowed is clearly wasting space, since many graphs in the population will be considerably smaller, and not have the maximum number of nodes.
4.2.2 Adjacency List

"The adjacency-list representation is quite robust in that it can be modified to support many other graph variants" ([3], p.466). This advantage should be kept in mind when problems are examined that require a different type of graph - and also if the project is extended to breed Petri nets.

A disadvantage of this representation is that the search for a node in the adjacency list requires traversing the whole list in the worst case, and thus comes with a time penalty.

4.2.3 Incidence Matrix

This graph representation, as described in [13], is also flexible as it can represent graphs with multiple edges and loops. At $O(VE)$ the space requirements are even higher than for the adjacency matrix - and the disadvantages that come with the adjacency matrix also apply to the incidence matrix.

4.2.4 Direct Representation with Explicit pointers

In this representation each node object has a set of edge objects associated with it. Each edge object points to (references) another node, and may also have a weight associated with it.

The disadvantage of this data structure is that even simple operations, like adding an edge, become challenging, since there is no direct access to the vertices. I don't think that this representation is suitable for the project described in this paper.

4.3 Genetic Operators for Graphs

During the course of the project this subsection will be filled with more details about genetic operators.

4.3.1 Mutation for Graphs

A mutation operator could modify graphs in the following ways:

1. change the weight of an edge
2. add a node
3. delete a node
4. add an edge
5. delete an edge

Compare these suggestions with the mutation operators suggested for finite-state machines in [1], p.247.

4.3.2 Recombination for Graphs

In analogy to [1], p.107, (crossover for tree structures) the major step will be the design of a recombination operator, which would simply and easily allow the creation of an offspring graph, using as inputs 2 graphs of generally high fitness as parents.

Conceptually, the crossover operation for graphs will be more complicated than that for tree structures. As stated on [1], p.156, "An appealing aspect of the parse tree representation is its natural recursive definition, which allows for dynamically sized structures." Achieving dynamically sized structures for graphs seems to be desirable too. For practical purposes a limitation policy that restricts the number of nodes or the number of edges in the graph needs to be introduced.

Summary of ideas:

- try to reuse ideas from [1], p.143,144 where matrix representations of permutations (TSP tours) are used. Check the work of Fox and McMahon, referenced on p.145 of [1], for matrix crossovers. (Here the number of nodes in the graph is a constant.)
- just union two (simple) graphs, as described in [13], p.453
- crossover at a set of nodes in some other fashion
- crossover at a set of edges

The goal must be not to disrupt good structures of subgraphs during the recombination process. Ideally, we want achieve a high chance that by the recombination process the combination of good (structural) building blocks leads to a better overall fitness of the graph.

Compare these suggestions with the recombination operators suggested for finite-state machines in [1], p.284-286; in particular Birgmeier's article looks interesting.

Also, Radcliffe's recommendations (reference given in [1], p.236) for properties that recombination operators should have to be useful could lead to new ideas.
5 Interpretation of an optimal solution

Once a satisfactory graph is found, one could try to study it, and try to figure out why the evolved graph performs better than one carefully designed by hand.

For the purpose of inspecting an evolved solution the following example was pointed out to me. Danny Hillis evolved sorting algorithms in [5] using simple swap operators. However they did not reveal any secrets as to how they work.

Another example is Genetic Programming (GP) - here programs are evolved that work, but for human beings it is almost impossible to understand their structure and thus how they work. In [6], p. 241, Koza evolved trigonometric identities and found the solution he obtained "opaque and incomprehensible". Still, he examines the solution further. There are exceptions to this, and they are also presented in [6], p. 184, where the solution to the 11-multiplexer problem contained a subexpression that could be identified as a 6-multiplexer.

As a philosophical side, evolution seems to produce objects, which, while adapted well to their current environment, are often structurally very complex, and at the same time they contain a good amount of randomness. Thus they are often elusive to human interpretation. Humans seem to prefer order, symmetry and simplicity when they design objects.

6 Petri Net Overview

The rest of this proposal is not directly related to the goals of this project - EC using graph representation. Instead its focus lies on the more distant research goal of breeding Petri nets - the evolution-directed search of the space of possible PNs for ones, which, when executed, will produce the best fitness. This section gives a brief introduction to Petri nets and mentions some fundamental publications in that field.

6.1 The Origin of Petri Nets

Nets are named for Prof. Dr. Carl Adam Petri, University of Hamburg, Germany, who showed in his dissertation in 1962 that there is a need for a theory of asynchronously working machines. (The English translation of his dissertation is [10].) At that time the term "process" had not been established yet in the area of computer science, but Petri’s schemata allowed the modelling of parallel and independent events in an illustrative manner.
His method turned out to be very useful when designing distributed systems
and processes and by 1968 it had become increasingly popular all over the
world. As a consequence his formalism was referred to as "Petri Nets".
This also sparked the emergence of a new research area within the branch
of theoretical computer science known as general net theory, which is concerned
with the laws of information flow.

6.2 Place/Transition Nets

References to Petri nets are [11] and [7]. The following descriptions use their
notation.

Definition 6.1 (Net)

A triple $N = (S, T, F)$ is called a net iff

1. $S$ and $T$ are disjoint sets and

2. $F \subseteq (S \times T) \cup (T \times S)$ is a binary relation called the flow relation of

   $N$.

In graph theoretical terms a net is a directed bipartite graph.

The following definition characterizes a rather simple class of low level
Petri nets. Very often when the term "Petri net" is used, it refers to
place/transition nets, which are defined as follows.

Definition 6.2 (Place/Transition Net)

A 6-tuple $N = (S, T, F, K, M, W)$ is called a place/transition net iff

1. $(S, F, T)$ is a finite net,

2. $K : S \rightarrow N \cup \{\omega\}$ gives a (possibly unlimited) capacity for each place,

3. $W : F \rightarrow N - \{0\}$ attaches a weight to each arc of the net, and

4. $M : S \rightarrow N \cup \{\omega\}$ is the initial marking where all the capacities are

   respected, i.e. $M(s) \leq K(s)$ for all $s \in S$.

Elements of $S$ are called places, elements of $T$ are called transitions.

Graphically, places will be represented by circles; transitions will be rep-
resented by squares. If $p \in S$ is a place, then the fact that $p$ contains $k$
tokens is represented by writing the number $k$ inside $p$'s circle.

This only defines the syntactic structure of a place/transition net, but
not yet its semantic properties. See figure 1 for a graphical overview of
place/transition nets.
Places look like: $\bullet$ K=7

The capacity of this place is 7 tokens.

Transitions look like:

An arrow from $p$ to $t$ with weight 4 indicates that $t$ needs 4 tokens from place $p$ to be able to fire.

An arrow from $t$ to $p$ with weight 3 indicates that $p$ gets 3 additional tokens after $t$ fires.

A place that contains 5 tokens looks like: 5

Figure 1: Place/Transition Nets Summary

The following definition helps to define the semantic properties of place/transition nets.

**Definition 6.3**

Let $N = (S, T, F)$ be a net. For $x \in S \cup T$

- $x_\bullet = \{ y | (y, x) \in F \}$ is called the preset of $x$,
- $x_{\bullet\bullet} = \{ y | (x, y) \in F \}$ is called the postset of $x$.

Now the semantics of place/transition nets can be defined.

**Definition 6.4**

Let $N = (S, T, F, K, M, W)$ be a place/transition net. A transition $t \in T$ is activated if

1. for every place $p$ in the preset $\bullet t$ the number of tokens at $p$ is greater or equal to the weight $W(p, t)$ of the arrow from $p$ to $t$ and
2. for every place $p$ in the postset $t_{\bullet\bullet}$ the number of tokens at $p$ plus the weight $W(t, p)$ of the arrow from $t$ to $p$ does not exceed the capacity $K(p)$ of $p$.  

13
Definition 6.5
Let $N = (S, T, F, K, M, W)$ be a place/transition net. An activated transition $t \in T$ may fire by

1. decreasing the number of tokens for all $p \in \bullet t$ by $W(p, t)$ and
2. increasing the number of tokens for all $p \in t \bullet$ by $W(p, t)$.

See [7], p.543, for illustrations of the firing rule.

6.3 Application areas of Petri nets

[7], p.542, lists (among many others) the following application areas:

- Communication protocols
- Performance evaluation
- Discrete event systems
- Manufacturing/industrial control systems
- Decision models
- Data flow computing systems

6.4 Summary of Petri net properties

"Petri nets are a promising tool for describing and studying information processing systems that are characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic." [7], p.541.

Advantages of Petri nets are:

- High illustrative potential.
- Relatively easy to understand
- The underlying graphical syntax is precisely defined
- Certain properties of a Petri net can be proved formally.
- There exists an underlying mathematical theory of an enormous size (most Petri net literature deals with that theory).
- Petri Nets are supported by software. In 1998 a study [14] evaluated existing Petri Net tools and came up with a top ten list out of 91 candidates.
• Concurrency features

[12] mentions the following additional properties of Petri nets:

• Petri nets emphasize locality of causation and effect.

• Petri nets are neither state-based nor action-based. Both states and actions have a particular status on their own.

• Petri nets explicitly represent fundamental issues of distributed systems, such as atomicity, synchronization, mutual independence of actions, messages, and shared memory.

Disadvantages of Petri Nets are:

• Petri nets don’t exhibit compositionality properties; i.e. merging two nets $N_1$ and $N_2$ that are correct for two different subproblems to form a net $N$ that is correct for both subproblems is a nontrivial task (see [8], p.39). This could become the major obstacle for using genetic recombination operators in the course of evolutionary computation.

• Petri nets suffer from a lack of abstraction potential. E.g. you cannot just leave out a certain part of an arbitrary design that you want to treat as a ”black box”, unless you adopt nonstandard conventions. Only under special conditions it is possible to have macro nodes that represent a subnet.

• Many tests procedures (e.g. matrix based methods) that check a certain Petri net property are static in nature. They ignore the execution order in which the rules/transitions fire. For some problems that might just be what is desired. But for other problems this execution order might be of particular interest. This shows that the representation chosen for the individuals of the population will depend on the problem to be solved.

7 The Idea of Evolving Petri Nets

Why might it be promising to evolve Petri nets? In this section I list several thoughts about this fundamental question.

PNs are a nice formalism that allow the modeling of certain real word problems. Their strength is their ”built in” concurrency mechanism, which allows a formal description of systems containing parallel processes. This
concurrency feature (transitions can fire in parallel) makes the PN a good model for problems, in which the order of certain actions does not matter.

Are there interesting problems, whose solutions can be represented as Petri nets, or not? The literature mentions numerous application areas for PNs. In particular, the design of industrial production systems can be represented with PNs. Optimizing such a system corresponds to optimizing the design of the associated PN. Existing PN theory and algorithms are able to answer many questions about formal characteristics of a PN. These theoretical results could be built into the fitness function. PNs with poor characteristics would score a low fitness value. Finding interesting problems, whose solution can be represented as a PN, should be an rather easy task.

From an operational perspective, there exist numerous software packages (see [14]) that can simulate the dynamic behavior of PNs ("Running the PN"). By observing the dynamic properties of the PN, a fitness value could be assigned to an individual PN. However the details of this are not completely clear to me yet. A PN does not directly return an output. The dynamical behavior is characterized by the sequence of the firing transitions. The stational behavior (the state) is characterized by the topology of the PN. Both the dynamic behavior and the static structure should influence the fitness function, I think. Another complication is that, given a Petri net and a marking, its dynamic behavior is not determined deterministically. There are many different firing sequences possible. This nondeterministic behavior might result in fitness differences for each simulation run. A fitness measure could be defined by taking the mean value of a sample of simulation runs.

In [9], one of the few references that employ the concepts of EC and PN together, PNs are used to limit the GA search to only legal individuals in a highly constrained scheduling problem. I found it to be an interesting approach to the ever present question of what to do with the many illegal individuals produced by GA, if the solution space is highly constrained.

Appropriate genetic operators need to be defined for PNs. Citing from [1], p.155, "When an executable structure such as a program or a function is the object of an evolutionary computation, representation plays a crucial role in determining the ultimate fitness of the system".

\subsection{Genetic Operators for Petri Nets}

"... the operators of any evolutionary system must be chosen carefully in accordance with the selected representation of individuals" [1], p.235.
7.1.1 Mutation for Petri Nets

A mutation operator could modify PNs in the following way:

1. change the weight of an arc
2. change the flow relation $F$
3. add a place or a transition or an arc
4. delete a place or a transition or an arc
5. change the capacity of a place
6. change the initial marking

Compare these suggestions with the mutation operators suggested for finite-state machines in [1], p.247.

7.1.2 Recombination for Petri Nets

The recombination operator for Petri nets would be a specialized version of a recombination operator for graphs. Since structurally Petri nets are directed bipartite graphs, the recombination operator for graphs needs to be tailored for Petri nets. The goal must be not to disrupt the behavior of certain sub-PNs during the recombination process. Then there is a chance that the combination of good (structural) building blocks leads to a good overall (dynamic) behavior of the Petri net.

I came up with the following basic crossover concepts:

- crossover at a set of places
- crossover at a set of transitions
- crossover at a set of arcs

Compare these suggestions with the recombination operators suggested for finite-state machines in [1], p.284.
References


