An Evolutionary Graph Transformation System as a Modelling Framework for Evolutionary Algorithms*

Hauke Tönnies

University of Bremen, Department of Computer Science
P.O.Box 330440, D-28334 Bremen, Germany
hatoe@informatik.uni-bremen.de

Abstract. In this paper an heuristic method for the solving of complex optimization problems is presented which is inspired equally by genetic algorithms and graph transformation. In short it can be described as a genetic algorithm where the individuals (encoding solutions of the given problem) are always graphs and the operators to create new individuals are provided by graph transformation. As a case study this method is used to solve the independent set problem.

1 Introduction

Solving combinatorial optimization problems, especially those whose decision variant belong to the complexity class of NP-complete, remains an important and highly active field of research, partly because of their importance in many industrial areas, partly because of the still unclear relationship between optimization problems that are NP-complete and optimization problems that are known to be solvable in polynomial time. A broad and diverse number of important optimization problems are NP-complete which means that in most cases realistic sizes of problem instances can only be solved by heuristic methods since an important property of NP-complete problems is that the worst case time behaviour of the currently best algorithm to solve them is exponential in the size of the problem instance.

This paper proposes to use graphs and graph transformation as a rule-based formal framework for modelling evoluionary algorithms which finally leads to a system called evolutionary graph transformation system. Graphs are quite generic data structure and therefore suitable for the modelling of a lot of interesting and complex optimization problems. The common search algorithms on graphs traverse the graphs to find either an optimal or an optimal-close solution, depending on the complexity of the problem. Graph transformation

* The author would like to acknowledge that his research is partially supported by the Collaborative Research Centre 637 (Autonomous Cooperating Logistic Processes: A Paradigm Shift and Its Limitations) funded by the German Research Foundation (DFG).

works directly on the representation of the problem, i.e. the graph, and alter the graph in a way that it represents a new and maybe better solution of the problem. Since graph transformation rules perfom local changes on graphs, it is a straight-forward idea, to model a mutation operator with graph transformation. Altogether with a suitable fitness and selection function and some rules for the initialization of the graphs, a nice way of modelling evolutionary algorithms is achieved.

The paper is organized as follows. In section 2 graphs and graph transformation are formally introduced. In section 3 it is shown how an evolutionary algorithm can be modelled by an evolutionary graph transformation system to solve the maximum independet set problem. The conclusion is given in section 5.

2 Graphs and Graph Transformation for Modelling static and dynamic systems

In this paper, undirected, edge-labelled graphs are assumed, although the presented method is suitable for any kind of graphs. In the following, we present some formal definitions:

Let V be some set. Then $\binom{V}{k}$ denotes the set of all subsets of V containing exactly k elements. Moreover, V^{1+2} is a notational shorthand for the union $\binom{V}{1} \cup \binom{V}{2}$.

Definition 1 (*Undirected, edge-labelled graph*)

Let Σ be an arbitrary set of labels. An undirected, edge-labelled graph is a quadruple $G = (V, E, att, l)$, where

1. *V is the set of nodes,*
2. *E is the set of edges,*
3. *$att : E \to V^{1+2}$ and*
4. *$l : E \to \Sigma$ are total functions.*

An edge e with $att(e) = \{v\}$ for one $v \in V$ is called *loop*. The sets V and E and the functions att and l of a graph G will be denoted $V(G)$, $E(G)$, att_G, l_G respectively. To cover unlabeled graphs a special element $* \in \Sigma$ is assumed, which is not drawn in the visualization of graphs. The set of all graphs over Σ is denoted by $\mathcal{G}_\Sigma$.

Definition 2 (*Subgraph*)

Let G and G' be two graphs. G is subgraph of G', if $V(G) \subseteq V(G')$ and $E(G) \subseteq E(G')$, $att(e) = att'(e)$ and $l(e) = l'(e)$ for all $e \in E(G)$.

Given a graph, a subgraph is obtained by removing some nodes and edges subject to the condition that the removal of a node is accompanied by the removal of all its incident edges. Let G be a graph and $X = (V(X), E(X)) \subseteq (V, E)$ be a

pair of sets of nodes and edges. Then $G - X = (V - V(X), E - E(X), att, l)$ is only then properly defined, if the above condition is met.

Definition 3 (*Graphmorphism*)

Let G and G' be two graphs. A graph morphism $\alpha : G \to G'$ is a pair of functions $\alpha_V : V^{1+2} \to V^{1+2}$ and $\alpha_E : E(G) \to E(G')$ that are structure-preserving.

The image $g(G)$ of a graphmorphism $\alpha : G \to G'$ in G' is called *match* of G in G' and is subgraph of G'.

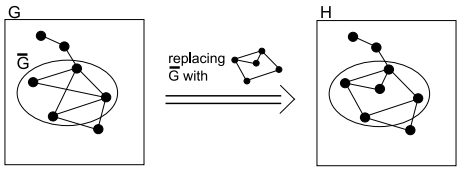

Fig. 1. Replacing subgraph $\overline{\text{G}}$ by a new graph

2.1 Graph Transformation

The idea of graph transformation consists in replacing a subgraph of a given graph by another graph (see Fig. 1). Usually this replacement is based on rules, the so called *graph transformation rules*.

Definition 4 (*Graph transformation rule*)

A graph transformation rule r *consists of three graphs $L, K, R \in \mathcal{G}_\Sigma$ with $L \supseteq K \subseteq R$.*

The graph to transform is called *hostgraph*. An application of a graph transformation rule r on a hostgraph G consists of three steps:

1. A graph morphism $\alpha : L \to G$ is chosen respecting two *application conditions:*
   (a) *Contact condition:* The removal of the match $\alpha(L)$ in G must not leave edges without attachment.
   (b) *Identification condition:* If two nodes or edges of L are identified in the match $\alpha(L)$ they must be in K.
2. The match $\alpha(L)$ is removed up to $\alpha(K)$ from G yielding a new graph $Z = G - (\alpha(L) - \alpha(K))$.
3. The graph R is added to Z by gluing Z with R in $\alpha(K)$ yielding the graph $H = Z + (V(R) - V(K), E(R) - E(K), att', l')$ with $att'(e') = att_R(e')$, if $att(e') \in (V(R) - V(K))^{1+2}$ and $att'(e') = \alpha_V(att_R(e'))$ otherwise and $l'(e') = l_R(e')$ for all $e' \in E(R) - E(K)$.

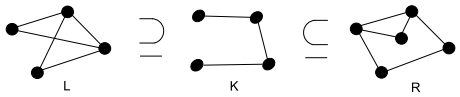

Fig. 2. Graph transformation rule

Analyzing in this respect the replacement of Fig. 1, the corresponding graph transformation rule is easy to obtain. The graphs L and R are obviously the graphs H and the graph depicted above the arrow in the figure. The graph K helps to unambigously embed the graph R in the hostgraph after the removal of $\alpha(L) - \alpha(K)$. That means, all the vertices and edges of L that we still need, should be in K. This leads to the rule depicted in Fig. 2.

The application of this rule on the host graph G yields the graph G'. In general, the application of a rule r to a graph G is denoted by $G \underset{r}{\Longrightarrow} G'$ and is called a direct derivation. A sequence $G_0 \underset{r_1}{\Longrightarrow} G_1 \underset{r_2}{\Longrightarrow} \ldots \underset{r_n}{\Longrightarrow} G_n$ of direct derivations can also be denoted by $G_0 \overset{n}{\underset{P}{\Longrightarrow}} G_n$, if $r_1, \ldots, r_n \in P$ or $G \overset{*}{\underset{r}{\Longrightarrow}} G'$ for $G = G_0$ and $G' = G_n$. The string $r_1, \ldots, r_n$ is called *application sequence* of the derivation $G_0 \underset{r}{\Longrightarrow} G_1 \underset{r}{\Longrightarrow} \ldots \underset{r}{\Longrightarrow} G_n$.

Graph transformation comprises a broad and wide area of application, for an overview see for example [1] . In this paper we focus on the application in the area of operations research. Before we show how graph transformation can be used to model algorithms to solve problems, we have to introduce one more important concept: the *graph transformation unit*. A graph transformation unit uses subsets of graphs specified by so called graph *class expressions* and so called *control conditions* to cut down the derivation process.

Definition 5 (*Graph class expression*)

A graph class expression is a syntactical entity X that specifies a set of graphs $SEM(X) \subseteq \mathcal{G}_\Sigma$.

A popular example of a graph class expression is a set of labels $\Delta \subseteq \Sigma$ such that

$$SEM(\Delta) = \{G \in \mathcal{G}_\Sigma \mid l(e) = \Delta \quad \forall e \in E(G)\}$$

Definition 6 (*Control condition*)

A control condition is a syntactical entity C that specifies a language $L(C)$ such that an application sequence s is permitted if and only if $s \in L(C)$.

Control conditions are useful to reduce the inherent non-determinism of applications of graph transformation rules. Without control conditions, any rule of the set P can be applied arbitrarily at any time, but sometimes it is preferable to control for example the order of the rule applications. In this case, regular expressions serve as an useful control conditions: Let $r_1 r_2^* r_3$ be a regular expression

as a control condition. All application sequences must then start with applying the rule r_1, following the application of an arbitrarily number of times rule r_2 and ending with the application of rule r_3. Another useful control condition consists in applying a rule *as long as possible* before other rules can be applied. To denote this control condition, we write an exclamation mark after the rule (e.g. $r_1 r_2 ! r_3$). More about graph transformation units and further examples of graph class expressions and control conditions can be found in [2], [3], [4].

Now we have all the components to define formally a graph transformation unit:

Definition 7 (*Graph transformation unit*)

A graph transformation unit is quadruple $tu = (I, P, C, T)$ where I, T are graph class expressions, P a set of rules and C a control condition.

A graph transformation unit defines a binary relation on the set of all graphs $\mathcal{G}_\Sigma$. Intuitivly, a graph G is tu-related to G', denoted by $G \underset{tu}{\Longrightarrow} G'$, iff $G \in SEM(I), G' \in SEM(T)$ and $G \overset{*}{\underset{P}{\Longrightarrow}} G'$ respecting the control condition C. As it is shown in the following sections, graph transformation units are useful in modelling algorithms on graphs.

Sometimes it is useful not only to transform a single graph, but a set or a multiset of graphs. Let tu be a transformation unit and $M : \mathcal{G}_\Sigma \to \mathbb{N}$ a multiset of graphs. A multiset transformation $M \underset{tu}{\Longrightarrow} M'$ transform all graphs of M by applying the transformation unit tu on all graphs of M. The multiset M' consists then of all transformed graphs. More about graph multiset transformation can be found in [5].

3 Solving Combinatorial Optimization Problems with Evolutionary Graph Transformation Systems

As mentioned before, the following heuristic needs the combinatorial optimization problem to be represented as a graph. Keeping in mind that most of the interesting combinatorial problems are modelled as graphs anyways, this restriction can be seen as an advantage to avoid finding an appropriate coding. The key idea of this method consists of a so called evolutionary graph transformation system that uses graph transformation units as a mutation operator to create new graphs which encode new solutions to the given problem.

3.1 An Evolutionary Graph Transformation System for the Independent Set Problem

First of all, a formal definition of an evolutionary graph transformation system is given.

Definition 8 (*Evolutionary graph transformation system*)

Let Σ be a set of labels and $\mathcal{G}_\Sigma$ the set of all graphs over Σ according to definition 1. An evolutionary graph transformation system is a system

$$evoGTS = (init, mut, fitness, selection, T, n)$$

where

1. init *is a graph transformation unit,*
2. mut *is a graph transformation unit,*
3. $fitness : \mathcal{G}_\Sigma \to \mathbb{R}$ *is a total function,*
4. $selection : \mathcal{M}_{finite}(\mathcal{G}_\Sigma) \to \mathcal{M}_{finite}(\mathcal{G}_\Sigma)$ *is a total function, where* $\mathcal{M}_{finite}(\mathcal{G}_\Sigma)$ *denotes the set of all finite multisets of* $\mathcal{G}_\Sigma$*,*
5. $T : \mathcal{M}_{finite}(\mathcal{G}_\Sigma)^* \to BOOL$ *is a total function and*
6. $n \in \mathbb{N}$ *a natural number.*

It is now shown, how these components interplay to solve a given optimization problem. As a running example a maximum independent set in the given graph is searched.

Fig. 3. Initiation rule

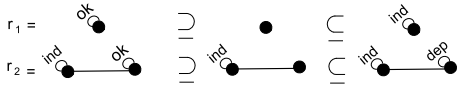

Fig. 4. Mutation rules

3.2 Solving Problems

1. The first step consists of initializing the set POP_0 through the transformation unit *init*.
$$POP_0 \underset{init}{\Longrightarrow} POP_1 \tag{1}$$
In this case, all graphs are initialized by putting an unlabeled loop to every node. The corresponding rule is depicted in Fig. 3. The control condition consists of applying this rule an arbitrary number of times and the terminal graphs, that is the succesfully initialized graphs, are exactly the graphs where every node has exactly one loop labeled with *OK*.
2. Once the multiset is initialized, the transformation unit *mut* is used to create a new multiset, which we will call, in analogy to other genetic algorithms, *children*.
$$POP_1 \underset{mut}{\Longrightarrow} children_1 \tag{2}$$

In case of the independent set problem a mutation consists of putting one random node, which has not been marked before, into the current independent set. The adjacent nodes are marked, so that the constraints of an independent set are never violated. The corresponding rule is depicted in Fig. 4 and the control condition is $(r1)(r2)!$.

3. Out of the two multisets POP_1 and $children_1$ the function *selection* chooses some graphs to form the multiset POP_2(the new parent generation) usually based on the fitness values. It seems a good idea to use well-known selection functions from other evolutionary algorithm.

$$selection(POP_1 + children_1) = POP_2 \tag{3}$$

One suitable fitness function for the independent set could be the following: Let S be the current independent set of a graph G.

$$fitness(G) = \begin{cases} \infty & if\ S = \varnothing \\ \frac{\sum_{s \in S} grad(s)}{|S|} & otherwise \end{cases}$$

4. It is checked, whether the function $T(POP_1, POP_2)$ yields *true*. If it does, the best graphs (e.g. the graphs with the lowest $fitness$-value) from POP_2 are returned. If it does not, the procedure beginning from step 2 starts again, using the multiset POP_2 as the multiset to be mutated.

In Fig. 5 a possible run of the evolutionary graph transformation system is shown, where the maximum independent set can be found in only one of the graphs in the last population.

4 Conclusion

In this paper, an evolutionary graph transformation system as a modelling framework for evolutionary algorithms has been proposed. In particular, it was shown how an evolutionary graph transformation system for the independent set problem could be modelled, which has turned out suitable to heuristically solve the problem. There is a broad and wide field for future research, so only a few examples are mentioned here. Future work could also extend the presented evolutionary graph transformation system, e.g. by providing a recombination operator or by allowing the simultaneous application of a set of rules and thus achieving parallelism. Furthermore, multi-criteria optimization can easily introduced by adjusting the fitness-function ([6]). Besides further theoretical investigation, this and other case studies should be implemented with one of the existing graph transformation system, e.g. GrGen ([7]) to gain some benchmarks and to prove the practical usefulness of modelling evolutionary algorithms with evolutionary graph transformation systems.

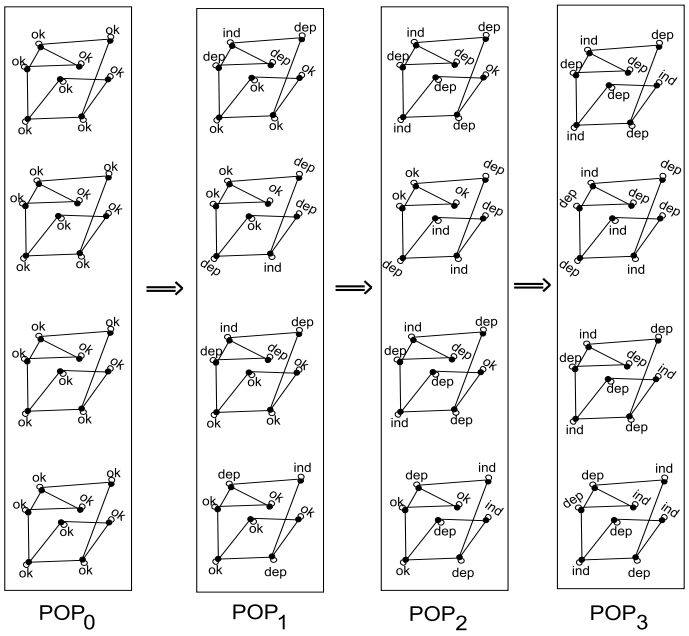

Fig. 5. Searching the maximum independent set

References

1. Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G., eds.: Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 2: Applications, Languages and Tools. World Scientific, Singapore (1999)
2. Kreowski, H.J., Kuske, S., Rozenberg, G.: Graph transformation units - an overview. In Degano, P., Nicola, R.D., Meseguer, J., eds.: Concurrency, Graphs and Models. Volume 5065 of Lecture Notes in Computer Science. (2008) 57–75
3. Kreowski, H.J., Kuske, S.: Graph transformation units with interleaving semantics. Formal Aspects of Computing **11**(6) (1999) 690–723
4. Kuske, S.: Transformation Units—A Structuring Principle for Graph Transformation Systems. PhD thesis, University of Bremen (2000)
5. Kreowski, H.J., Kuske, S.: Graph multiset transformation as a framework for massively parallel computation. In: Proc. 4th Intl. Conference on Graph Transformations (ICGT 2008). Volume 5214 of Lecture Notes in Computer Science. (2008) 351–365
6. Deb, K.: Multi-Objective Optimization Using Evolutionary Algorithms. John Wiley & Sons, Inc., New York, NY, USA (2001)
7. Geiß, R., Kroll, M.: GrGen.NET: A fast, expressive, and general purpose graph rewrite tool. In Schürr, A., Nagl, M., Zündorf, A., eds.: Proc. 3rd Intl. Workshop on Applications of Graph Transformation with Industrial Relevance (AGTIVE '07). Volume 5088 of Lecture Notes in Computer Science. (2008) 568–569