TRANSPORT NETWORK OPTIMIZIATION: SELF-ORGANIZATION BY GENETIC PROGRAMMING

J. Göbel[(a)], A.E. Krzesinski[(b)], B. Page[(a)]

[(a)] Department of Informatics, University of Hamburg, 22527 Hamburg, Germany
[(b)] Department of Mathematical Sciences, University of Stellenbosch, 7600 Stellenbosch, South Africa

[(a)]{goebel,page}@informatik.uni-hamburg.de, [(b)]aek1@cs.sun.ac.za

ABSTRACT

This goal of the paper is transport network optimization. *Transport networks* are defined as network topologies where entities are forwarded from node to node constrained by capacity restrictions both on nodes and links. Examples include urban traffic (vehicles/signalized intersections) and IP networks (packets/routers). Optimization of such networks particularly has to provide the logic the nodes use to determine which entity to process next. Such logic can be imposed by a central authority based on global knowledge of the network state. In contrast, a *self-organizing* network solely relies on local decision rules to prioritize entities. At the cost of a potential loss in performance, such a decentralized network control is scalable and robust. This paper proposes *genetic programming* to evolve local node rules. Results indicate that the performance is similar to centrally (near-optimally) controlled systems.

Keywords: transport network optimization, genetic programming, discrete event simulation, simulation framework, Java

1. INTRODUCTION

The target of the work is the optimization of a general class of networks, namely *transport networks*. These are defined as graph topologies formed by nodes and links; continuously, entities "appear" at originating nodes (all nodes or only a subset may qualify as originating nodes). Such entities are queued for being "processed" by their origin nodes. After processing, the entities traverse links, thus queuing for processing at the next nodes on their routes until eventually reaching their destination nodes. Examples of such networks include urban traffic (vehicles advancing from one intersection to the next), conveyor-based manufacturing systems (items processed successively by different workstations) and telecommunication networks (e.g. IP packets being forwarded from one router to the next).

Optimizing such a transport network typically may involve minimizing waiting or travel times or maximizing throughput. Apart from discarding entities or adjusting their routes (which may or may not be feasible, depending on the network type) and long-term improvements to the network topology itself (e.g. increasing nodes' capacities, establishing additional links), the only degree of freedom for achieving such targets is the logic the nodes use to determine which entity (e.g. vehicle, item, IP packet) to process next.

This logic for entity prioritization can be set up by a central authority, which is typically provided with global knowledge of the network state. For example, based on global (estimated) traffic density data, signals can be coordinated such that platoons of vehicles are able to traverse the network without stopping ("green waves"). However, such attempts to centrally optimize such networks typically imply exponential computational complexity (Holland 1995), yielding bad scaling behaviour. Further assuming the requirement to adaptively adjust to dynamic changes in traffic patterns, such approaches depend on the availability of the central server and the communication to this authority.

This motivates applying decentralized optimization: Without being dependent on a central authority, each node (router, workstation, traffic light) independently decides which entity is processed next. This decision is based on information that is locally available (e.g. queue lengths, local flow estimations) only, enabling nodes to act autonomously if assuming means of obtaining theses data (e.g. induction loops and cameras and image processing capabilities at an urban intersection).

Literature – e.g. in Bazzan 2005, Cools 2007, Gershenson 2005, Helbing 2008, Lämmer 2007 attempting decentralized urban traffic optimization – already provides local node control logic performing almost as well as or better than centrally controlled systems for some special network topologies, e.g. for Manhattan grids with one-way traffic (i.e. north to south and west to east traffic only, Gershenson 2005) or intersection inflows from different directions assumed to be mutually exclusive (Lämmer 2007).

The remainder of this paper is organized as follows: Section 2 provides further details about decentralized transport network optimization. Section 3 proposes genetic programming (GP) as potential solution, evaluated in Section 4 by experiments in a simulation environment. The paper concludes with a summary and outlook about further work in Section 5.

2. DECENTRALIZED TRANSPORT NETWORK OPTIMIZATION

Transport network optimization can be conducted by a central authority to which all relevant information is made available; examples include Diakaki 2003, Pohlmann 2010 or commercial systems like SCOOT (see e.g. United Kingdom Department for Transport 1995). However, apart from the dependence on the communication of each node to this central authority, the run-time performance of such approaches scales badly with the network size, compare Section 1. Furthermore, as optimization is typically conducted in cycles of 15-60 minutes, reaction to patterns of traffic shifting or the failure of an adjacent node is delayed.

This motivates applying decentralized optimization, analogously to the concept of *self-organization* in thermodynamic and other natural sciences: A *self-organizing* system autonomously acquires and maintains order despite external influence subject to perturbations, which is typically achieved by a set of microscopic (local) decision rules independently used by each component (Wolf 2005). However, the development of such rules if not known in advance is difficult; designing a self-organizing system can be interpreted as reverse-engineering such rules from the desired macroscopic behaviour of the system: This behaviour (e.g. efficient transportation) is an emergent property of such rules, for which research thus far does not offer an agreed-upon and universally applicable means of obtaining (Zambonelli 2004).

Nonetheless, the development of such local rules facilitating decentralized optimization of a transport network can be approached from the input side, i.e. the information locally available at the nodes: From Bazzan 2005, Cools 2007, Gershenson 2005, Helbing 2008, Lämmer 2007 and other approaches to solve special cases of the network optimization problem, a set of criteria can be obtained, upon which the decision as to which entities to prioritize at a given instant is based.

Using from now on urban traffic terminology for example, these criteria apply either to a specific lane (a queue for vehicles arriving at a node on a certain link, waiting for processing and departure on one or more other links), to an intersection as whole (e.g. maximum queue length, total estimated arrival rate), or even to the overall network (e.g. switching penalty, during which all lanes are "red" for safety reasons). See Table 1 for further examples for such criteria.

Lämmer 2007 has also shown that any node logic can be expressed as function which he refers to as the *priority index*: Based on a subset of these criteria, one can determine the priority index of each lane; serve the vehicles from the lane with the highest priority, unless network stability would be violated if a lane did not receive any service for some maximum waiting period. Table 2 shows an extension of this mechanism using a set of lanes instead of single lanes, taking into account intersections serving more than one lane simultaneously (e.g. opposing traffic from north and south proceeding straight ahead).

Flow (lane)
Queue length (F, LQL)
Longest waiting time (F, LWT)
Est. arrival rate overall (F, LAO)
Est. arrival rate current period (F, LAP)
Current phase since (F, LPS)
No green since (F, LGS)
Utilization of incoming link (F, LIU)
Maximum utilization of all outgoing links (F, LOU)
Est. arrivals next period (F, LEA)
Est. duration until next platoon arrival (F, LPA)
Est. feasible flow/absolute (F, LFA)
Est. feasible flow/relative (F, LFR)

Node (intersection)
Max. queue length (F, NQL)
Longest waiting time (F, NWT)
Est. arrival rate/overall (F, NAO)
Est. arrival rate/current period (F,NAP)
Duration of current phase (F, NPL)
Idle (B, NID)
blocked (B, NBL)

Global (network)
Average acceleration (F, GAA)
Switching penalty (F, GSP)
Passing possible in queues (B, GPP)

Table 1: Examples of criteria for entity (vehicle) prioritization in urban traffic; the suffix states the data types, either floating point numbers (F) or Boolean (B), and an abbreviation.

Lane set selection can for example be conducted on *greedy* basis: The lane with the highest priority is set to green. Repeatedly, from all lanes not mutually exclusive to a lane already set to green, the lane with the highest priority is set to green until no further lane exists that is neither set to green nor mutually exclusive to a lane already set to green.

Determine priority index for each lane
Unblock set of lanes with highest accumulated priority
Repeatedly re-evaluate priority indices and switch to a different set of lane once their accumulated priority exceeds the priority of the current flows by a positive Δ
Ensure minimum green and maximum waiting periods are not violated

Table 2: Link choice based on priority indices, extended from Lämmer 2007

With the *input* (the above-mentioned criteria) and the *result* (a priority index function) being specified, the open problem is determining how to obtain the latter from the former. For special network topologies, this

problem is already solved (compare literature quoted above), but not for the general case.

Part of a more general solution might be setting local rules such that certain desirable patterns – like green waves – are facilitated, see Göbel 2009. However, for network configurations, in which no such desirable patterns are known or where patterns available do not suffice for fully specifying a priority index function, we propose a different approach based on *genetic programming* (GP) in Section 3.

3. NODE LOGIC EVOLTION BASED ON GENETIC PROGRAMMING

Our target is to obtain priority index functions for decentralized transport network control based on the input criteria from Table 1. In the absence of any restrictions of which of these criteria to use and how to algebraically and logically determine a priority index (PI) from their respective values, genetic programming, see e.g. Koza 1992 or Poli 2008, has been chosen because is a flexible and robust search technique not relying on any preconditions which would limit is applicability to special cases of transport networks: Mimicking nature, programs to determine priority indices are evolved by successively improving existing programs. Note that GP can be interpreted as a generalization of genetic algorithms (GA): While the search space of a typical GA is a static chromosome on which parameter values to optimize are stored, this structure the chromosome in GP itself is subject to evolution.

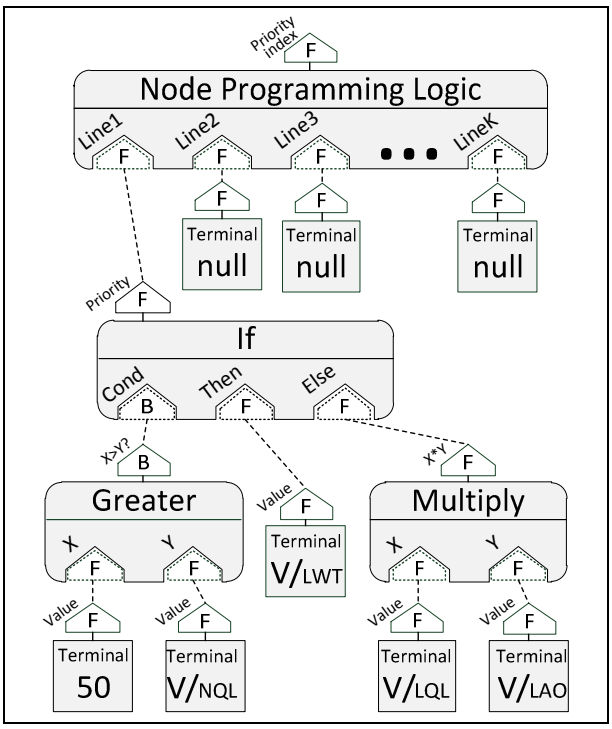

Figure 1: Example of a GP node priority index function

To facilitate the evolution of genetic programs of arbitrary complexity, genetic programs typically have a tree-like structure consisting of nodes ("building blocks") representing functions that can be flexibly combined, with the only restriction that arguments and result data types of adjacent node functions have to match.

An example priority index function is shown in Figure 1, composed of functions like for instance "Multiply" (bottom right), representing a function accepting two floating point numbers (F) as arguments and returning the product as floating point number as result. The overall result of the GP program is the return type of the top-level function ("Node programming logic"), which itself requires arguments determined by functions on the second level and so forth. Note that the closure of genetic programs assumes the availability of functions not requiring any argument, so-called terminals, e.g. numerical or Boolean constants or values read from variables (notation V/… to distinguish from constant terminals), like the node's maximum queue length (V/NQL), the longest waiting time of an entity at a lane (V/LWT), a lane's queue length (V/LQL) or a lane's estimated overall arrival rate (V/LAO), compare Table 1.

A typical genetic programming cycle is summarized by Figure 2: Based on an initial population of programs generated at random, so-called genetic operators are applied to simulate biological evolution: The fitter (i.e. better network control performance) a program, the higher its probability of being selected for offspring composition by means of recombination; Figure 3 shows examples of recombination operators, particularly chromosome transfer ❶, aggregation ❷, projection ❸, and swapping ❹.

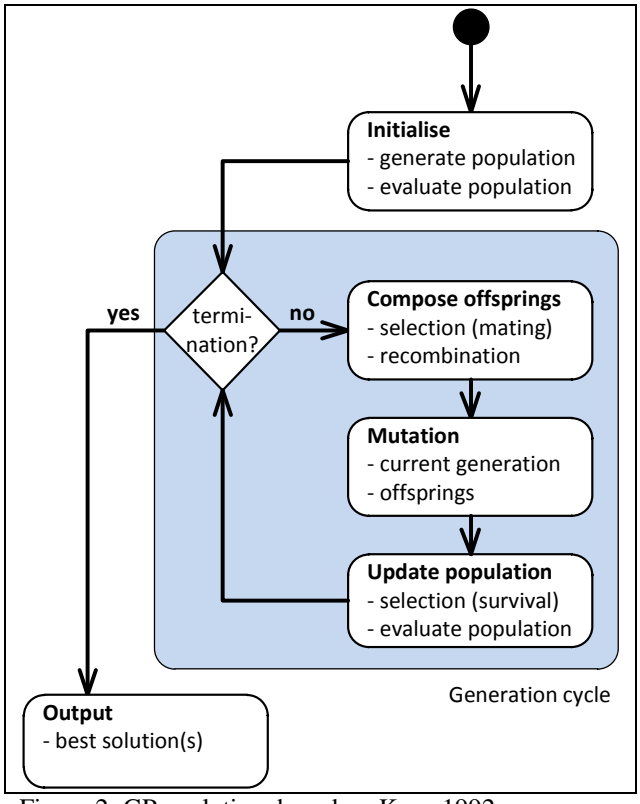

Figure 2: GP evolution, based on Koza 1992

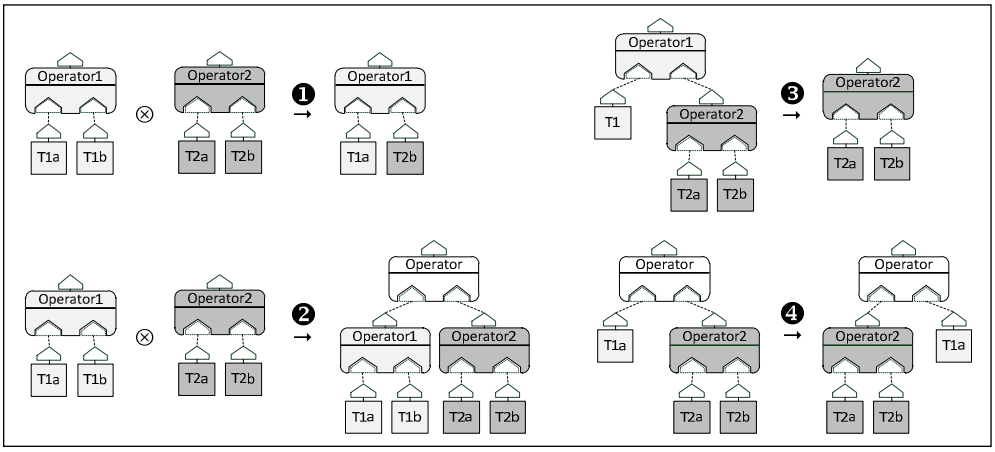

Figure 3: GP recombination operators

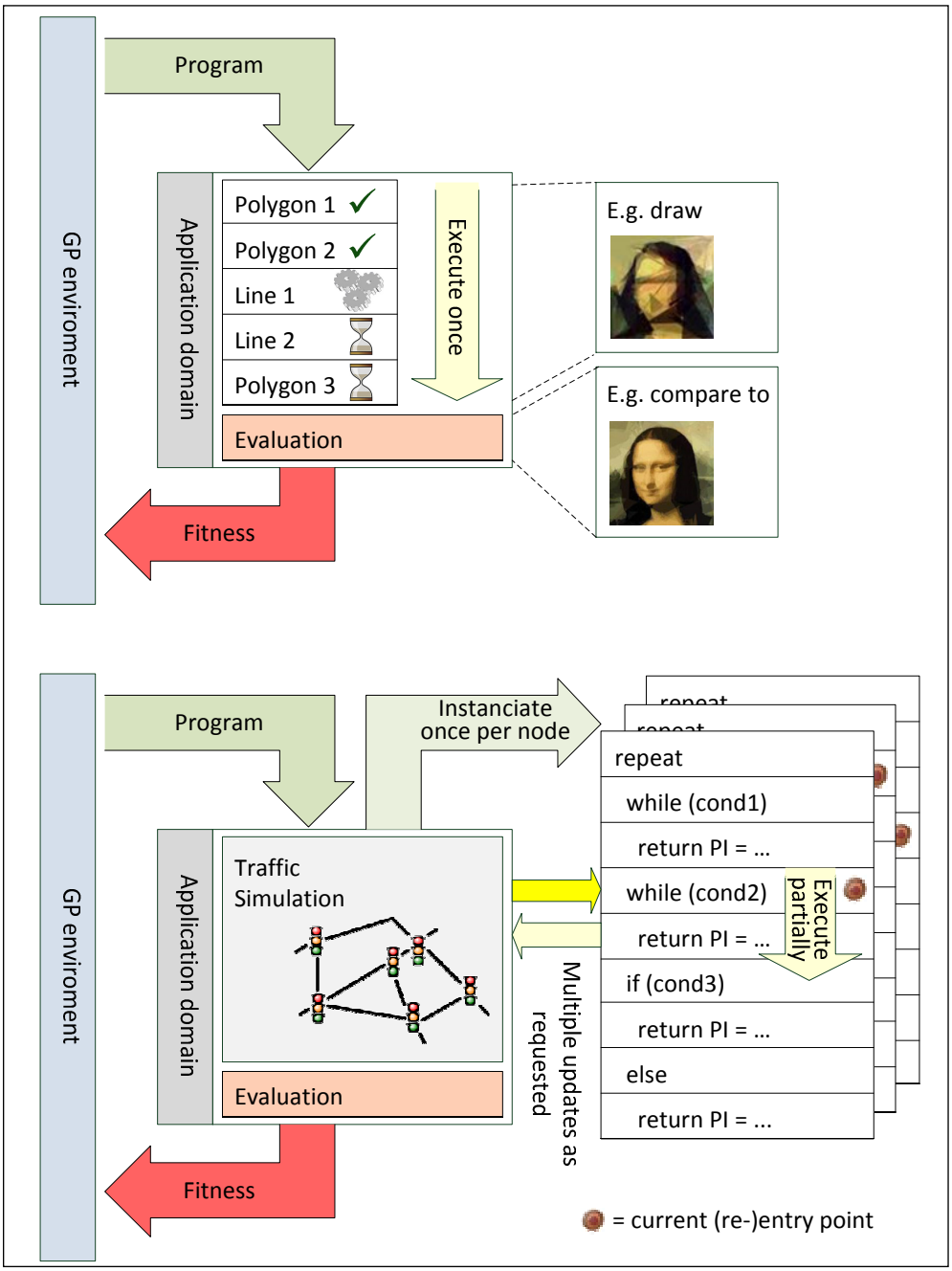

Figure 4: Alternative means of evaluating the fitness of a program

Both existing programs and new offsprings have a small chance of undergoing a random mutation, e.g. functions replaced by other functions with the same input and result parameter types, including terminals potentially being replaced by other terminals of the same type. For the resulting set of programs, a fixed number is selected to "survive" and form the next generation.

The set of functions typically used to GP-based mathematical functions includes numerical (plus, minus, multiply, divide, power, root, exp, log, abs) and logical operators (and, or, xor, not, greater, smaller) as well as statements to define piecewise functions (if/else, switch).

Inversion of control and state-dependent priority indices

The standard GP approach of determining the fitness of a program is shown in Figure 4, upper half: The full program – e.g. serving the purpose of creating a Mona Lisa forgery by drawing lines and filled polygons on a canvas (courtesy of Meffert 2011) – is executed once, followed by evaluating the fitness by comparison to the "real" Mona Lisa.

For two reasons, this paradigm is not appropriate for determining node priority index (PI) functions:

First, the evaluation of a PI function is driven by a simulation experiment calling the PI function whenever desired.

Secondly, local PI functions as proposed by Gershenson 2005 or Lämmer 2007 are dependent on previous calls to the PI functions since they use partial conditionals (e.g. "if" without "else", leaving PI calculation to the next line if the condition does not apply) and loops. Allowing such program components potentially yields programs evaluated only partially, compare Figure 4, bottom half: An equation determining the PI inside the "while"-function for instance may be evaluated repeatedly until the "while"-condition is no longer fulfilled.

Our implementation, based on JGAP (Java Genetic Algorithms Package, see Meffert 2011) reflects this control flow. Particularly, we propose a special function referred to as "Node Programming Logic" (see also Figure 1), which keeps a reference to the current (re-)entry point, i.e. the "line" to use at the moment to determine the priority of a lane, thus emulating program execution. Note that multiple nodes in a network require multiple program instances with a different (re-)entry point token each to reflect nodes potentially being in different states. The return type of "Node Programming Logic" is a PI floating point value, which allows for for hierarchically nesting partial conditionals and loops using multiple "Node Programming Logic" functions. After all lines have been used for determining a set of lanes to set to green, the program execution resumes at the first line.

Wrapping up, the program from Figure 1 will assign the highest priority to the lane of the vehicle with longest waiting time, which yields a FIFO (First in, first out) service, see the "then" branch of the "if" clause, as long as congestion is moderate (less than 50 entities queued in total). Otherwise, lane priority is the product of queue length and arrival rate: As congestion increases, the function tends to prefer main roads and to serve multiple vehicles before switching to other links.

Section 4 will investigate the performance of this GP approach of decentralized network optimization for three example networks (one intersection, two intersections, a small city area).

4. EVALUATION

We have built a discrete event simulation environment which is sufficiently parametrizable to represent different kinds of transport networks like urban traffic and IP packet routing; see Göbel 2009 for details about this environment. The mesoscopic logic of entity movement is derived from the traffic queuing model proposed by Nagel 2003. This simulation is based on DESMO-J, a framework for discrete event modelling and simulation in Java (see Page 2005 and the web page at http://www.desmoj.de), developed at the University of Hamburg.

To evaluate the performance of the described GP approach of transport network optimization, we have investigated three network scenarios S1, S2, S3:

- S1 consists of a single isolated intersection with incoming traffic from two directions with identical conditions (same speed limit of 50 km/h, single lanes).
- In S2, two intersections are located 100 meters apart. Symmetrical traffic flow is restricted to W→N and E→S, yielding mutual exclusiveness at both intersections (see Figure 5, assuming right-hand traffic).
- S3 is a network consisting of 11 intersections from southern Hanover/Germany subject to various flows from almost any entry to almost any exit, see Pohlmann 2010.

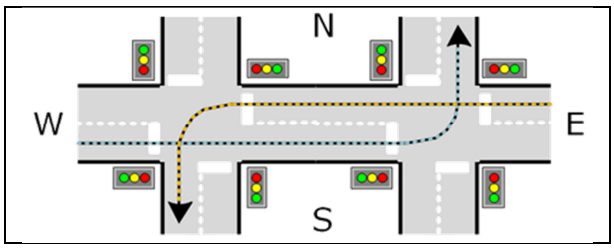

Figure 5: Scenario 2 (Two intersections)

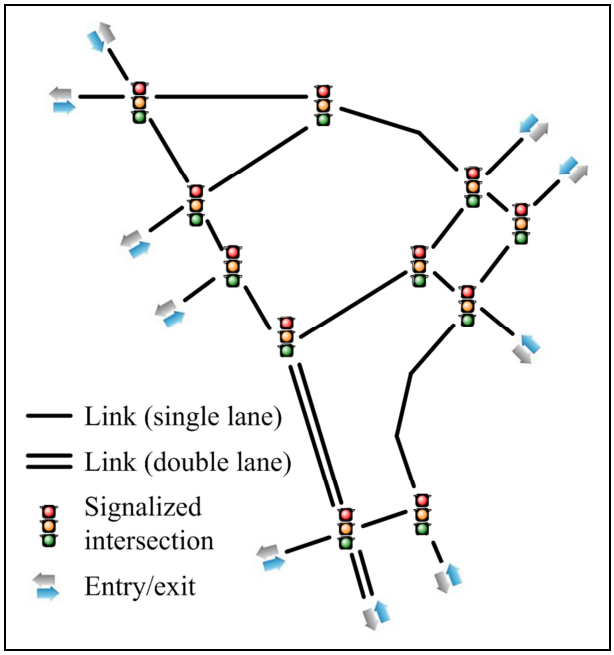

Figure 6: Scenario 3 (Hanover)

The optimization target is to minimize the vehicles' average overall waiting times. For S1, a near-optimal strategy is alternatingly ("round robin") serving each flow until the queue is empty: Switching earlier is not optimal as intersection capacity would be given away due to the switching penalty incurred in terms of a two second safety period in which all traffic lights have to be red; switching later most likely wastes capacity as no vehicle is served (unless the next arrival of a vehicle not yet queued is very close at hand).

Since the space between the intersections is limited in S2, the optimization problem particularly involves synchronizing their traffic lights such that both intersections never waste capacity by being unable to server either of the flows. This undesired situation

occurs if the link towards the other intersection is fully congested (thus no further incoming vehicles from W at the western intersection and from E at the eastern intersection can be served) while at the same time no vehicles bound for N/S already served by the other intersection are waiting for service. A near-optimal centralized solution for S2 is "fill and clear", exploiting the symmetry of the traffic flows offered: At both intersections the incoming flows from W at the western intersection and from E at the eastern intersection receive green until the link between the intersections is filled or until both queues incoming are empty. Synchronously, the vehicles bound for N/S now receive green at both intersections until the link between the intersections is cleared. Assuming enough "supply" of incoming vehicles, neither of the intersections ever wastes capacity apart from symmetry deficits caused by stochastic noise, e.g. one intersection clearing its vehicle queue on the link between the intersections faster than the other.

For S3, a heuristic is used serving the longest queues (typically four protected flows on a four-way intersection, e.g. W→S, E, N and N→W in right-hand traffic) until the queues for different combinations of flows are least 25 vehicles longer, yet at least for 5 seconds.

Network	Control	Avg. Wait	Through-put
S1 (low load)	Round robin	5.7	3167
	GP	5.4	3161
S1 (high load)	Round robin	17.4	6243
	GP	19.0	6085
S1 (overload)	Round robin	141.6	6376
	GP	135.5	6459
S2	Fill and clear	13.6	5170
	GP	13.0	5201
S3	Longest queue	77.3	18663
	GP	52.9	23377

Table 3: Experiment performance results

Network	Program

S1/S2	repeat
	  if (V/NID)
	  return PI = (V/LIU)
	  else
	  return PI = exp(V/LFA)

S3	repeat
	  if (V/NID)
	  return PI = V/LAO
	  else
	  return PI = exp(V/LPF) * V/LWT
	  while (N/NQL > 68.4445)
	  return PI = V/GAA*exp(exp(V/LQL))

Table 4: GP evaluation results

Table 3 compares these means of intersection control to the best GP solution found in 100 generations of size 100 for S1/S2 (combined) and for S3; the table states average waiting time and vehicle throughput during 5 hours (average of 10 runs). S1 has been evaluated with three different load levels (approx. 3200, 6400 and 9200 vehicles offered). The GP fitness function was the average waiting duration (the lower, the better), subject to a penalty proportional to the node-count of the genetic program, thus implicitly bounding the complexity the programs evolved. Table 4 shows the "fittest" programs found in each of the runs S1/S2 and S3.

Comparing the results of GP to the (near-)optimal solutions in the case of S1 and S2 or to the heuristic in S3 yields a GP performance similar or better (with the exception of the second S1 load level): Although not applying centralized control, e.g. explicit traffic light synchronization in S2, the GP solutions perform approximately equally well or in some cases even slightly better by exploiting the marginal remaining optimization potential, e.g. asymmetrical link clearance in S2 used to advance the traffic light switch at the relevant intersection which is advantageous in terms of overall waiting durations if the flow set to green is slower than its counterpart receiving green later.

5. SUMMARY AND OUTLOOK

This paper has presents a GP-based approach to decentralized, transport network optimization, providing local rules in terms of priority index functions. To the standard paradigm of GP evolution (Koza 1992), adjustments were necessary to cover inversion of control in fitness evaluation (simulation calling the program to be evaluated, not vice versa) and state-dependently only partially executing the program to be evaluated; these adjustment were implemented extending JGAP (Java Genetic Algorithms Package, see Meffert 2011). Experiment results indicate that the performance is similar to centrally (near-optimally) controlled systems while at the same node control is scalable and not dependent on a central authority. "Performance" of course is not restricted to minimizing waiting times as conducted in the experiments in Section 4; the GP-based transport network is sufficiently flexible to use any fitness function, e.g. a weighted combination of waiting times and fuel consumption/emission production.

Further work will address the run-time performance of the GP evolution of node PI functions: As the fitness evaluation of a single program is relatively expensive due to the discrete event simulation runs to be executed (approx. 1 day for scenarios S1/S2, approx. 4 days for scenario S3 on a single machine), recognizing and not evaluating inferior programs may provide large improvements in run-time performance. Examples of such inferior programs are all programs containing branches that are never executed (e.g. all lines after a while(true) {...} statement) or programs not containing a single lane-specific criterion (compare Table 1) as they yield the same priority for all lanes at an intersection.

The convergence of the GP can also be improved by providing "higher level" criteria, e.g. including the

optimal priority for an isolated intersection subject to uniform flows (no stochastic noise) as determined by Lämmer 2007, thus relieving the GP evolution from producing such terms.

Another means of facilitating GP convergence is removing the need for co-evolution by allowing a sub-tree referenced more than once: If the results determined by a certain sub-tree, the current GP approach would be required to create this repeating pattern more than once (Figure 7, left). Allowing multiple references (Figure 7, right) has to ensure infinite recursion is avoided, yet provides smaller programs without need to multiple branches undergoing the same evolution.

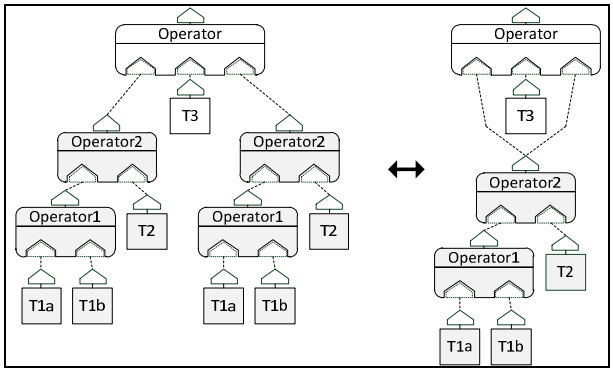

Figure 7: Multiple references to a sub-tree

REFERENCES

Bazzan, A., Oliveira, D. de, and Lesser, V., 2005. Using Cooperative Mediation to Coordinate Traffic Lights: A Case Study. *Proceedings of Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 463-469, July 25-29, Utrecht (The Netherlands).

Cools, S.-B., Gershenson, C., and D'Hooghe, B., 2007. Self-organizing traffic lights: A realistic simulation. In M. Prokopenko (ed): *Self-Organization: Applied Multi-Agent Systems*, pp. 41-49. London (UK): Springer.

Diakaki, C., Dinopoulou, V, Aboudolas, K., Papageorgiou, M., Ben-Shabat, E., Seider, E., and Leibov, A., 2003. Extensions and new applications of the traffic signal control strategy TUC. *Transportation Research Board*, 1856:202-211.

Gershenson, C., 2005. Self-Organizing Traffic Lights. *Complex Systems* 16(1):29-53.

Göbel, J., 2009. On Self-Organizing Transport Networks – an Outline. *Proceedings of the 6th Vienna International Conference on Mathematical Modelling (MATHMOD) 2009*, p. 82. Feb 11-13, Vienna (Austria).

Helbing, D., and Lämmer, S., 2008. Self-Control of Traffic Lights and Vehicle Flows in Urban Road Networks. *Journal of Statistical Mechanics: Theory and Experiment* 4(P04019):1-33

Holland, J. H., 1995. *Hidden Order – How Adaption builds complexity*. New York (New York, USA): Basic Books.

Koza, J. R., 1992. *On the programming of computers by means of natural selection*. Cambridge (Massachusetts, USA): MIT Press.

Lämmer, S., 2007. *Reglerentwurf zur dezentralen Online-Steuerung von Lichtsignalanlagen in Straßennetzwerken*. PhD Thesis, Technical University of Dresden (Germany).

Meffert, K. et al., 2011: *JGAP – Java Genetic Algorithms and Genetic Programming Package*. URL: http://jgap.sf.net

Nagel, K., 2003. Traffic networks. In S. Bornholdt, H. G. Schuster (eds): *Handbook on networks*. New York (NY, USA): Wiley.

Page, B. and Kreutzer, W., 2005. *The Java Simulation Handbook – Simulating Discrete Event Systems with UML and Java*. Aachen (Germany): Shaker.

Pohlmann, T., 2010. *New Approaches for Online Control of Urban Traffic Signal Systems*. PhD Thesis, Technical University of Braunschweig (Germany).

Poli, R., Langdon, W. B. and McPhee, N.F., 2008. *A Field Guide to Genetic Programming*, Raleigh (North Carolina, USA): Lulu.com

United Kingdom Department for Transport, 1995. "SCOOT" Urban Traffic Control System. *United Kingdom Department for Transport Traffic Advisory Leaflet* 04/1995.

Wolf, T. de and Holvoet, T., 2005. Emergence Versus Self-Organisation. In S. A. Brueckner, et al. (eds): *Engineering Self-Organising Systems*, pp. 1-15. Berlin (Germany): Springer.

Zambonelli, F., Gleizes, M.-P., Mamei, M., and Tolksdorf, R., 2004. Spray Computers: Frontiers of Self-Organization. *Proceedings of the First IEEE International Conference on Autonomic Computing (ICAC'04)*, pp. 268-269, May, Miami (Florida, USA).

AUTHORS BIOGRAPHY

J. Göbel holds a diploma in Information Systems from the University of Hamburg, Germany. He is scientific assistant and PhD candidate at the Center of Architecture and Design of IT-Systems at the University of Hamburg; his research interests focus on discrete event simulation and network optimization.

A. E. Krzesinski obtained the MSc from the University of Cape Town and the PhD from Cambridge University, England. He is a Professor of Computer Science at the University of Stellenbosch, South Africa. His research interests centre on the performance evaluation of communication networks.

B. Page holds degrees in Applied Computer Science from the Technical University of Berlin, Germany, and from Stanford University, USA. As professor for Applied Computer Science at the University of Hamburg he researches and teaches in the field of Modelling and Simulation as well as in Environmental Informatics.