Evolving Control Laws for a Network of Traffic Signals

David J. Montana and Steven Czerwinski
BBN Systems and Technologies
70 Fawcett Street, Cambridge, MA 02138
dmontana@bbn.com, czerwin@mit.edu

Abstract

Optimally controlling the timings of traffic signals within a network of intersections is a difficult but important problem. Because the traffic signals need to coordinate their behavior to achieve the common goal of optimizing traffic flow through the network, this is a problem in collective intelligence. We apply a hybrid of a genetic algorithm and strongly typed genetic programming (STGP) to the problem of learning control laws which optimize aggregate performance. STGP learns the single basic decision tree to be executed by all the intersections when deciding whether to change the phase of the traffic signal. The genetic algorithm learns different constants to be used in these decision trees for different intersections, hence allowing specialization based on differences in geometry and traffic flow. Preliminary experimental work shows that our approach yields good performance on a variety of network configurations and that it can evolve control laws which induce cooperation, communication, and specialization among the traffic signals.

1 Background

1.1 Networks of Traffic Signals

Most traffic signals in use today use preset timings. These preset timings are optimized off-line for the traffic patterns at a particular time of day. At predetermined times during the day, a new set of timings is downloaded to each traffic signal in the network. Two popular packages for doing the off-line optimization are MAXBAND [Little *et al.*, 1981] and TRANSYT-7F [FHWA, 1986].

For the purposes of this paper, we are interested in a different, newer type of control known as adaptive traffic signal control. This type of control takes into consideration the variations in traffic flow (both variations around the average and variations from the average) which naturally occur. Based on sensory inputs of the current traffic conditions, the control algorithm modifies the signal timings over time.

SCOOT [Robertson and Bretherton, 1991] starts with a set of preset timings as optimized by TRANSYT. At fixed intervals it modifies the splits, offsets, and cycle times of each signal so as to try to improve the performance evaluation of this signal and its immediate neighbors. OPACS [Gartner *et al.*, 1991] also has each signal modify its timing to optimize some local evaluation criterion; its major innovation is the use of dynamic programming as a way to efficiently look further ahead in its predictions of future performance. SCATS [Sims, 1979] and Chiu's fuzzy logic approch [Chiu, 1992] both use heuristic rules as a way to adjust signal timings; the major difference is that Chiu's rules are fuzzy rules.

A shortcoming of all these techniques is that a potential adaptation is evaluated based only on its effect on a small neighborhood of signals over a small interval of time. The global effects of this adaptation on the full network are ignored despite the fact that these effects are often significant. Hence, for our work we draw on an area of research that focuses on how to coordinate individual actions to produce desired aggregate behavior.

1.2 Collective Intelligence

The concept of *collective intelligence* has been developed in the fields of Robotics and Artificial Life. The basic idea is that many individual agents each executing simple, deterministic behaviors can interact in such a way that the aggregate behavior of the group is complex and not fully predictable from knowledge of the individual behaviors. One early example of artificial collective intelligence is the Boids simulation, which simulates flocking of birds [Reynolds, 1987]. A variety of work has followed, including some using real robots, e.g. [Mataric, 1994].

One important issue in collective intelligence and in our work is that of communication [Arkin and Hobbs, 1993]. Explicit communication can aid coordination among agents, but there generally is a cost (in the case of traffic signals a cost which is measurable as the price of the extra wires and extra interfaces required).

Another important issue in collective intelligence, and the key one for our work, is learning the control strategies which allow agents to cooperatively perform a task. For learning control laws for a group of robots, [Mataric, 1994] uses a type of reinforcement learning. For learning control of a colony of ants foraging for food, [Collins and Jefferson, 1992] use a genetic algorithm to learn the weights and connectivity of an artificial neural network. Most relevant to our work are those approaches using genetic programming (GP) as the learning paradigm. Koza's work on learning strategies for ants foraging for food was the first to use GP in this context [Koza, 1990]. Subsequent work includes Ryan's use of GP to evolve teams of callback functions in an event-driven system [Ryan, 1995]. Most similar to our work is that of [Haynes *et al.*, 1995], which uses strongly typed GP (STGP) to evolve strategies for packs of predators trapping a prey.

1.3 Genetic Programming Enhancements

Strong Typing: Strongly typed GP (STGP) [Montana, 1995] is an enhancement to standard GP [Koza, 1992] where (i) all the functions know what data types they take as arguments and what data type they return, and (ii) the routines for tree generation, mutation, and crossover all ensure that data types are consistent. The big advantage of STGP over GP when using typed data is that STGP eliminates the huge overhead of having to evaluate syntactically incorrect parse trees (which greatly outnumber the syntactically correct parse trees).

GA/GP Hybrids: An issue in GP is how to generate constants for use in the parse trees. The standard approach is to define a terminal $\Re$ which generates a random numerical constant in some range each time it is initially placed in a tree [Koza, 1992]. This method works well when the constants required are single, isolated numbers. However, especially when using STGP, the constants required are often not single numbers but rather more complex data structures such as the matrices in [Andre, 1994]. Extending the terminal $\Re$ to generate random instances of these data structures makes the search for correct values for these constants essentially a random search, which is known to be impractically inefficient for higher-dimensional search. A more practical alternative is to utilize a hybrid GA/GP search [Andre, 1994; Nguyen and Huang, 1994]. The GA performs the search to find the right values for the constants, while the GP searches the space of parse trees. The chromosome representation contains both the parse tree and the constant values, and there are different genetic operators for manipulating the parse trees and the constants.

2 Our Approach

2.1 The Simulation

The simulator we are using is a special version of the TRAF-NETSIM traffic simulator. TRAF-NETSIM has been developed by the U. S. Federal Highway Administration and has been one of the standards of the traffic engineering field. It is a microsimulator, which means that it simulates individual vehicles rather than average flows. It allows the user to specify via a file the geometry of the network, the traffic flows, and the signal timings (which for the standard TRAF-NETSIM are fixed).

A special version of TRAF-NETSIM has been developed for evaluating adaptive traffic signal control. It contains two major enhancements. First, sensors can be placed in the road in order to detect the passage of vehicles. Second, the fixed signal timings have been replaced with hooks for a user-implemented adaptive control law. This control law utilizes the sensor outputs in determining when to change the signal phase. The control law is executed for each traffic signal once every second.

We have linked our enhanced genetic programming code with the simulator to allow execution of a specified parse tree as the main part of an adaptive control law. The high-level logic of this control law is as follows. We require that there be exactly four phases for the signal, two green phases and two corresponding yellow phases. (This simplifying assumption excludes situations such as delayed-green phases and left-turn-only phases.) The yellow phases all last some fixed amount of time, which for the experiments described below was five seconds. To determine when to end a green phase, we execute the parse tree every second. The parse tree returns a Boolean value which is true if the signal should change at the current time and false if it should not.

We require that there be two sensors for each link of an intersection. One should be placed right next to the intersection to count the cars as they enter the intersection. The other should be placed further away to count the cars as they approach the intersection. The placement of this second sensor should be such that the queue for that link is considered "full" if cars are stopped back to that sensor. In the experiments described below, we always placed this second sensor 120 feet from the signal.

2.2 Phase-Dependent Constants

A phase-dependent constant is a constant in a parse tree which has a different value for each phase of each intersection. A simple example of how a phase-dependent constant can be used is the parse tree (> SINCE-CHANGE CONSTANT-1), where SINCE-CHANGE is the time since the current signal last changed color and CONSTANT-1 is a phase-dependent constant. This tree implements fixed-timings control where the length of each phase is given by the value of CONSTANT-1 for that phase. Phase-dependent constants are useful because they provide a mechanism to have each phase of each intersection execute a single basic parse tree while allowing for the fact that each phase is constrained by its own particular geometry and traffic patterns and requires customization

Function	**Argument Types**	**Return Type**
NUMBER-CARS	PHASE	INTEGER
WAITING	PHASE	BOOLEAN
APPROACHING	PHASE	BOOLEAN
QUEUE-FULL	PHASE	BOOLEAN
OR	BOOLEAN BOOLEAN	BOOLEAN
AND	BOOLEAN BOOLEAN	BOOLEAN
NOT	BOOLEAN	BOOLEAN
>	INTEGER INTEGER	BOOLEAN
SINCE-CHANGE		INTEGER
CONSTANT-i		INTEGER
MOVING		PHASE
STOPPED		PHASE

Figure 1: Baseline functions with their types.

of its parse tree for these factors.

Because we have assumed that each intersection has exactly two phases, each phase-dependent constant will have $2N$ values, where N is the number of intersections. These $2N$ values can be thought of as a $2N$-valued real string, which can be evolved using a standard string-based genetic algorithm. To evolve phase-dependent constants simultaneously with the parse tree, we use a GA/GP hybrid as described in Section 1.3.

2.3 The Functions

Figure 1 shows the functions utilized in all the experiments described below. Note that the PHASE data type enumerates the $2N$ different green phases discussed in Section 2.2. We now describe these functions.

NUMBER-CARS examines all incoming links for the specified phase and returns the maximum number of cars between the two sensors for any link (calculated as the difference of the vehicle counts of the two sensors).

WAITING tells whether there is a car currently stopped waiting for the specified phase to turn green (as detected by there being a car sitting directly over the sensor closest to the intersection for any incoming link).

APPROACHING tells whether there is a car which is approaching the specified phase (as detected by the vehicle count of the second sensor of an incoming link exceeding that of the first).

QUEUE-FULL tells whether there are cars backed up to the second sensor for any incoming link of the specified phase (as detected by there being a car sitting directly over this second sensor).

SINCE-CHANGE returns the time since the current intersection (i.e., that intersection for which the control law is currently being executed) last changed its phase.

CONSTANT-1 and CONSTANT-2 are two phase-dependent constants that are simultaneously evolved along with the parse tree (see Section 2.2). They return the entry in the real-valued string which corresponds to the current green phase of the current intersection.

MOVING (STOPPED) returns the current green (red) phase of the current intersection.

Function	**Argument Types**	**Return Type**
DS-NUM-SAME	PHASE	INTEGER
DS-NUM-CROSS	PHASE	INTEGER
DS-WAIT-SAME	PHASE	BOOLEAN
DS-WAIT-CROSS	PHASE	BOOLEAN
DS-GREEN-SAME	PHASE	BOOLEAN
DS-GREEN-CROSS	PHASE	BOOLEAN

Figure 2: Additional communications functions

Figure 2 shows six additional functions used only in one of the experiments. We refer to these functions as "communications functions" because they require data from the sensors of neighboring intersections. In a real traffic network, implementing such functions would require extra expenses, e.g. additional wires being laid in the ground. Therefore, we consider these experimental functions whose potential contributations need to be determined first in simulation. The functions are:

DS-NUM-SAME returns the maximum number of cars between the two sensors for any link downstream from the specified phase.

DS-NUM-CROSS returns the maximum number of cars between the two sensors for any link crossing an intersection downstream from the specified phase.

DS-WAIT-SAME returns whether there is a car waiting at any link downstream from the specified phase.

DS-WAIT-CROSS returns whether there is a car waiting at any link crossing an intersection downstream from the specified phase.

DS-GREEN-SAME returns whether any link downstream from the current phase has a green signal at the downstream intersection.

DS-GREEN-CROSS returns whether any link crossing an intersection downstream from the current phase has a green signal at that intersection.

Note that all these communications functions address downstream links. This is because there is always implicit communications from the upstream links based on the number of cars approaching, while there is no such mechanism for implicit communication from the downstream links. Hence, communication from the downstream links is more likely to be useful.

2.4 The Evaluation Function

The basic measure of effectiveness we use for evaluating performance is delay. Delay is defined as the total amount of time lost due to the traffic signals. This in-

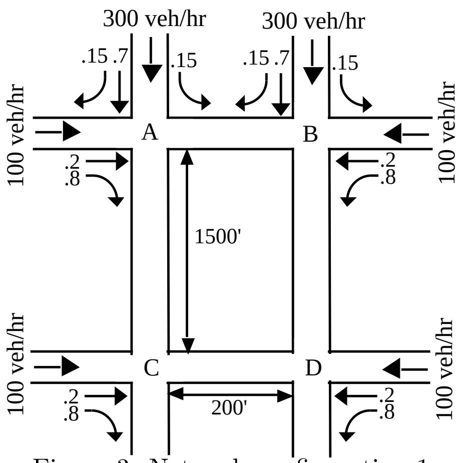

Figure 3: Network configuration 1.

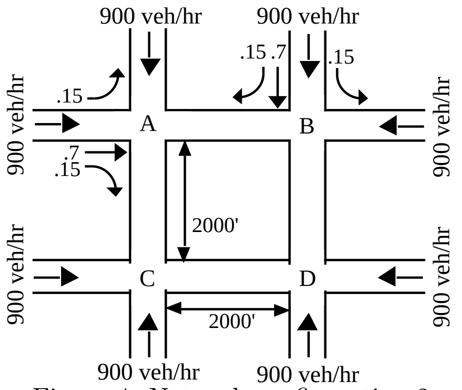

Figure 4: Network configuration 2.

cludes not just the time stopped at a traffic signal but also the time lost traveling at less than free-flow speed when decelerating to a stop and then accelerating again.

The precise criterion is the sum over each phase of each intersection of the number of cars passing this phase times the square of the average delay for this phase. This measure can be easily calculated from the statistics TRAF-NETSIM keeps and represents a reasonable tradeoff between efficiency and equity. By weighting the contribution by the number of cars, we make sure that we give more weight to the phases with larger number of cars, hence promoting efficiency. However, by using the square of the average delay rather than just the average delay, we ensure that the phases with smaller number of cars cannot be ignored, hence promoting equity.

3 Experimental Results

3.1 The Network Configurations

For the experiments we have performed so far, we have used three different network configurations. Figure 3 shows the first configuration. It contains two streets running north-south, both one-way in the south direction, and two two-way streets running east-west. Vehicles approaching an intersection from the east or west turn onto the one-way street with a probability of 0.8 and go straight with a probability of 0.2. Vehicles approaching an intersection from the north turn left with a probability of 0.15, turn right with a probability of 0.15, and go straight with a probability of 0.7.

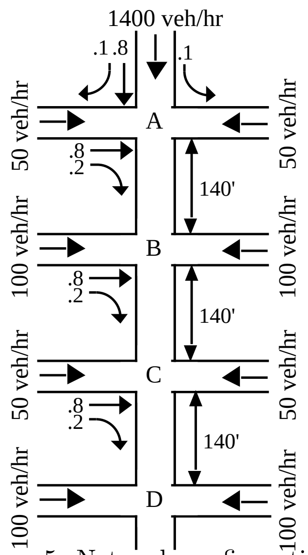

Figure 5: Network configuration 3.

There are six sources of vehicles entering the network. These sources randomly generate vehicles with an exponential distribution for the time between vehicles. The two sources on the north-south streets generate an average of 300 vehicles per hour, while the four on the east-west streets generate 100 vehicles per hour.

The intersections are arbitrarily labeled A-D. This ordering of the intersections is used when accessing the vector of values for a phase-dependent constant. The two phases of intersection A are the first two entries in this vector, then the two phases of intersection B, and so on. By convention, the north-south phase of an intersection precedes the east-west phase.

Figure 4 shows the second configuration. It contains two streets running north-south and two east-west, all two-way streets. The network is fully symmetric.

Figure 5 shows the third configuration. It contains one one-way street running north-south and four two-way streets running east-west.

Note that all three geometries contain four intersections. This was a good number for preliminary work, but to prove that our approach is practical requires that in future work we investigate larger networks and the scaling issues involved.

Also note that we have not yet considered a configuration where the average traffic flows vary with time. Since the ability to adjust for such variations is potentially the main benefit of adaptive control, experiments with time-varying flows are a high priority for future work.

Configuration	1	2	3
Population Size	1000	2000	5000
Evaluations	1000	2000	13000
Tree Score (Training)	22.62	540.7	20.13
Tree Score (Test)	21.81	558.0	20.99
Fixed Score (Training)	24.20	579.6	24.78
Fixed Score (Test)	24.83	640.2	28.76

Figure 6: Summary of experimental results.

3.2 The Results

For each configuration, we performed three experiments. The first experiment was to make a run with the approach described above to see what the best individual was and how well it performed. For all three configurations we used the functions in Figure 1, but only for the third configuration did we used the communications functions of Figure 2.

The second experiment was to use a genetic algorithm to optimize fixed-cycle timings to see how well a fixed-cycle approach could perform on the same problem. In this case, the individuals were real-valued strings of length twelve, three entries for each intersection. Two of the entries for each intersection were the phase splits (i.e., how long each phase stays green), while the third entry was the offset, which tells at what point in the cycle to start. The phase split entries could vary between 0 and 30, while the offset entries could vary between 0 and 99 (indicating the percentage of the cycle completed at the start). The population size was 1000 and the number of evaluations 10,000 for each run. The rationale for this experiment is to compare the adaptive control law we evolved with the best possible fixed-cycle control.

The third experiment was to make three runs with three different random seeds using the best individual from the STGP/GA run and then the best individual from the fixed cycles run. The rationale for this experiment is to see how well the two techniques generalize to test cases with the same geometry and flow statistics.

A summary of the runs is given in Figure 6. Note that these results clearly indicate that the adaptive control we evolve: (i) is superior to fixed-cycle control and (ii) generalizes well to new situations with the same statistics. Some other observations are the following. The good generalization of the trees was probably due to the simplicity of the trees. As more complex trees are required (for example, when we start looking at variations of the average flows), we expect generalization performance to drop unless we compensate with longer simulations. In two of the three cases, the best individual was found in the initial population, and evolution gave no benefit. This is often the case when using strong typing (which restricts the search space) and when the best tree is simple [Montana, 1995]. Finding more complex trees should require the full power of evolution rather than just a random search.

We now analyze the the best individual for each STGP/GA run. Note that we have reduced each of these individuals to their simplest form to simplify analysis.

Configuration 1: The best individual had tree

```
(OR (> CONSTANT-2 CONSTANT-1)
    (APPROACHING STOPPED))
```

and constants

```
Constant 1 = [ 27 10 11 16 14 12 20 13 ]
Constant 2 = [ 6 24 4 18 11 30 13 17 ]
```

Observe that the constant values for the north-south phases (the odd-numbered entries) are such that (> CONSTANT-2 CONSTANT-1) is false while for the east-west phases (even-numbered entries) (> CONSTANT-2 CONSTANT-1) is true. Hence, for the north-south phases the tree is equivalent to (APPROACHING STOPPED), while for the east-west phases the tree is always true. Therefore, the strategy is to always keep each signal green for the north-south street except when a car is detected approaching on the east-west street. This makes sense when we observe that (i) the traffic flows are light and (ii) the north-south streets are more heavily traveled than the east-west streets. Note that this is a good example of specialization using the phase-dependent constants.

Configuration 2: The best individual had tree

```
(AND (APPROACHING MOVING)
     (WAITING STOPPED))
```

This tree succeeds because it provides a good tradeoff between allowing vehicles which are moving to maintain their momentum and not forcing vehicles which are stopped to wait too long. It may seem counterintuitive to wait until a vehicle is approaching a green light before changing to yellow until we remember that the sensor is only 120 feet from the intersection. Hence, not only the vehicle which triggered the sensor but also one or two vehicles behind it can make it through the intersection before the signal turns red. Because this configuration has a moderately heavy traffic load, if there is not currently a vehicle approaching the signal, there soon will be, often as part of a small group. Keeping the signal green long enough to allow a group through allows maintenance of momentum. There is coordination of effort (i.e., cooperation) because each traffic signal tends to hold vehicles to form them into small groups, which another traffic signal can exploit.

Configuration 3: The best individual had tree

```
(AND (APPROACHING STOPPED)
     (NOT (DS-WAIT-SAME MOVING)))
```

Observe that (DS-WAIT-SAME MOVING) is always false for all the east-west phases and the north-south phase of the southernmost intersection. Hence, for these five phases, the tree is equivalent to (APPROACHING

STOPPED), which for the east-west phases is almost always true due to the large traffic flow on the north-south street. For the other three north-south phases, the term (NOT (DS-WAIT-SAME MOVING)) is for momentum preservation. The timing is such that keeping the phase green when the downstream intersection has cars waiting on the same street guarantees that the additional cars which are let through the intersection have no delay at the downstream intersection.

4 Conclusions and Future Work

The results are preliminary (due to the limited size and complexity of the problems) but encouraging. We were able to train our controller for a variety of geometric configurations and traffic conditions. The fact that our approach outperformed fixed timings shows that it is indeed adapting to variations around the average and hence fulfilling some of the promise of adaptive control. It was able to coordinate the individual traffic signals for aggregate goals, both with and without explicit communication. Finally, our approach allowed specialization, i.e. different phases of different intersections executing qualitatively and quantitatively different behaviors.

However, many key questions remain unanswered. These include: (i) how to make our approach handle a large number of intersections, (ii) how our approach handles variations in the average traffic flows, (iii) how our approach compares in performance with other adaptive control approaches, and (iv) how well a control strategy evolved for a simulation performs in a real network.

Acknowledgments

This work was supported under FHWA Grant DTFH61-94-X-00026. We would like to thank: Christina Andrews, who helped us get started with TRAF-NETSIM; and Alberto Castillo, who assisted with the experiments.

References

[Andre, 1994] D. Andre. Automatically defined features: The simultaneous evolution of 2-dimensional feature detectors and algorithm for using them. In K. Kinnear, editor, *Advances in Genetic Programming*, pages 477–494. MIT Press/Bradford Books, 1994.

[Arkin and Hobbs, 1993] R. C. Arkin and J. D. Hobbs. Dimensions of communications and social organization in multi-agent robotic systems. In *Proc. Second Int'l Conf. on Simulation of Adaptive Behavior*, pages 486–493, 1993.

[Chiu, 1992] S. Chiu. Adaptive traffic signal control using fuzzy logic. In *Proc. Intelligent Vehicles '92*, pages 98–107, 1992.

[Collins and Jefferson, 1992] R. J. Collins and D. R. Jefferson. AntFarm: Towards simulated evolution. In *Artificial Life III*, pages 579–601, 1992.

[FHWA, 1986] U. S. FHWA. TRANSYT-7F self-study guide, 1986.

[Gartner *et al.*, 1991] N. H. Gartner, P. J. Tarnoff, and C. M. Andrews. Evaluation of optimized policies for adaptive control strategy. *Transportation Research Record*, 1324:105–114, 1991.

[Haynes *et al.*, 1995] T. Haynes, R. Wainwright, and S. Sen. Strongly typed genetic programming in evolving cooperation strategies. In *Proc. Sixth Int'l Conf. on Genetic Algorithms*, pages 271–278, 1995.

[Koza, 1990] J. R. Koza. Genetic evolution and co-evolution of computer programs. In *Artificial Life II*, pages 603–629, 1990.

[Koza, 1992] J. R. Koza. *Genetic Programming.* MIT Press/Bradford Books, Cambridge, MA, 1992.

[Little *et al.*, 1981] J. Little, M. Kelson, and N. Gartner. MAXBAND: A program for setting signals on arteries and triangular networks. *Transportation Research Record*, 795:40–46, 1981.

[Mataric, 1994] M. J. Mataric. Learning to behave socially. In *Proc. Third Int'l Conf. on Simulation of Adaptive Behavior*, pages 453–462, 1994.

[Montana, 1995] D. J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.

[Nguyen and Huang, 1994] T. Nguyen and T. Huang. Evolvable 3D modeling for model-based object recognition systems. In K. Kinnear, editor, *Advances in Genetic Programming*, pages 459–475. MIT Press/Bradford Books, 1994.

[Reynolds, 1987] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.

[Robertson and Bretherton, 1991] D. Robertson and R. D. Bretherton. Optimizing networks of traffic signals in real time - the SCOOT method. *IEEE Trans. on Vehicular Tech.*, 40(1):11–15, 1991.

[Ryan, 1995] C. Ryan. GPRobots and GPTeams - competition, co-evolution, and co-operation in genetic programming. In *AAAI Symposium on Genetic Programming*, pages 86–93, 1995.

[Sims, 1979] A. Sims. The Sydney Coordinated Adaptive Traffic system. In *Proc. ASCE Conf. on Computer Control of Urban Traffic Systems*, pages 12–27, 1979.