TRAFCOD: A Method for Stream-Based Control of Actuated Traffic Signals

Peter G. Furth
Dept. of Civil and Environmental Engineering
Northeastern University
Boston, MA 02115

Theo H.J. Muller
Faculty of Civil Engineering and Geosciences
Delft University of Technology
Delft, The Netherlands

ABSTRACT: In the TRAFCOD method, control of traffic signals at an intersection is formulated in terms of traffic streams, without reference to stages (sets of streams that are green together). The resulting formulation is more flexible and direct than stage control, allowing for easy implementation of a variety of traffic control tactics, including priority tactics for public transport, demand responding realization and coordination between streams.

Control is expected to follow a given cyclic sequence structure; however, streams with no demand may be skipped, allowing the control program under some circumstances to uncouple into rings that cycle independently. Sequence state is formulated using an array of successor functions called the sequence array (for each conflicting stream pair, the array element is either 0 or 1 depending on which stream has the next turn). Changes in the sequence state are effected by rotations of the sequence array.

Control formulas are expressed in terms of standard traffic engineering inputs, allowing for automatic generation of control formulas. Related TRAFCOD software includes a program for automatic formula generation and a keyboard-actuated controller simulator used for training traffic engineering students. Tests confirm that the model works as expected.

Vehicle actuated intersection control means that the timing and, in more advanced controllers, the sequence of the signal indications are responsive to vehicle detections. While timing decisions are important, the focus of this paper is sequence control; in other words, if several conflicting streams have requests, which will become green next?

Flexibility in sequence control depends on the controller hardware and the software that drives the controller. Both controller hardware and software can be classified as either stream-based or stage-based. (A *stream* is a traffic stream with its own signal indication; a *stage* is a set of streams that are green simultaneously. More complete definitions follow in the body of the paper.) Stream-based hardware, which is standard in most of northern Europe including Great Britain *(1),* allows most of the

control logic to be formulated in user-programmable software that outputs to the controller the signal indications.

Compared to the traditional stage-based controller, stream-based controllers offer advantages for both actuated and fixed time control *(2, 3).* From a traffic engineering point of view, there are three main arguments for stream-based control. First, efficiency – clearance times can be enforced stream to stream rather than stage to stage, leading to less lost time. Second, simplicity – stages and overlaps need not be spelled out, but simply occur as a byproduct of how the streams follow each other. Third, flexibility – because most traffic control tactics relate directly to streams rather than to stages, stream-based control more easily allows the traffic engineer to apply desired control tactics. For example, minimum green time is meaningful for streams, but not for stages. Likewise, most tactics for safety, coordination, and priority are most easily formulated at the stream level. For example, with stream-based control, phase insertion becomes simply a matter of checking for conflicting streams, without the need to specify all the stages to which a priority stream could possibly belong. During periods of light traffic, stream-based control makes it easy to apply tactics for safety and efficiency such as skipping no-demand streams and waiting in red. The application of various safety tactics is often one of the main reasons for applying stream-based control in the Netherlands and in Scandinavia *(4).* Stream-based controllers have been called acyclic *(5)* because they don't require a cyclic service sequence, although they can be used with a cyclic sequence as well.

Of course, because a stage is a set of streams, the software driving a stream-based controller can still be formulated in terms of stages. Surprisingly, stage-based control appears to be the norm with vehicle-actuated signals not only in the United States, where stage-based hardware dominates, but also in some countries with stream-based controllers such as Germany *(6)* and Great Britain (including the MOVA control system) *(7).* While American double-ring controllers *(8)* and German control software allow some responsiveness in stage sequence, they are still less flexible than stream-based control.

The stream-based control language TRAFCOL (TRAFfic COntrol Language) and derivative products have been used in the Netherlands since the early 1970's. An abbreviated version (omitting administrative functions such as checking that detectors are working) called TRAFCOD (TRAFfic COntrol Design) *(9)* was developed in 1978 to train traffic engineering students. However, even though control languages used in the Netherlands are stream-based, their structure and / or application often place artificial restrictions on stream sequence logic. For example, one common restriction is to require that a fixed stage appear every cycle to provide a clear cycle start, regardless of demand, in order to simplify sequencing logic. Another popular software product *(10)* controls sequence by dividing the cycle sequence into modules, which are a kind of super-stage, with rules for allowing the program under various circumstances to cross module boundaries. While the rules that allow for such exceptions restore a good deal of flexibility, they are often not used to full advantage, depending on the expertise of the traffic engineer. Just as important, the ad hoc nature of the sequence logic in existing control methods usually requires that detailed specification of control formulas which rely on specialized knowledge and are prone to error.

Research on stream-based sequence control has recently led to a method, now incorporated in an updated version of the TRAFCOD language, that appears superior in several respects to other known sequence control methods. It is based on a very general model of sequence state that is extremely flexible, with no reference to stages, modules, or streams that must have green each cycle. With this model, most traffic control strategies, including various priority tactics for public transport, can be formulated in terms of basic traffic engineering data, allowing control formulas to be generated automatically. Instead of spending a lot of effort to make sure that control formulas are cleverly and correctly specified, the traffic engineer can instead concentrate on higher level decisions such as which traffic control tactics to use.

The following section clarifies the terminology of stream-based control. Next the new model of stream sequence used in TRAFCOD is explained. The main section of the paper lays out the principal TRAFCOD sequence control formulas, followed by some illustrative examples. The last sections describe additional control tactics and the status of ongoing work with TRAFCOD.

Streams, Stages, Phases, and Activation

A *traffic stream* or stream is the smallest unit of control in the intersection. Generally, it is the set of movements sharing the same queue and controlled by either a single signal or a set of signals that must always give the same indication.

Two streams are *conflicting* if they may not have the right of way simultaneously. A *clearance time matrix* expresses the minimum required interval between the start of stream *i*'s red and stream *j*'s green for each conflicting *i*,*j* pair. Clearance times depend on intersection geometry and stream speeds as well as local policy, and are therefore not generally symmetric.

Various types of coordination constraints may apply to non-conflicting stream pairs. *Simultaneous start* means that a pair of streams must start green at the same time, unless one is skipped for lack of demand. This tactic is mostly applied for safety reasons to opposing traffic streams with permitted left turns, in order to help establish priority for the through movements. It may be implemented with or without a *common request*, meaning that a request on one stream is taken as a request on the other. There are also various types of *staggered start* constraints, in which one stream's green is to start a certain interval after another's. Staggered start can be used for safety reasons to help establish priority (e.g., have a bicycle or pedestrian stream begin 2 s before a parallel general traffic stream with permitted right turns). It can also be used to provide a green wave for movements through two stop-lines such as pedestrians using a series of crosswalks, or vehicles at complex intersections.

A *stage* is a set of streams that receive the right-of-way simultaneously. Viewed globally, an intersection's signals can be seen as going from one stage to another. Elsewhere, stages are sometimes called phases.

However, we use the term *phase* to refer to the state of a stream's signal, the main phases thus being green, yellow, and red. In the Netherlands, the green phase is generally divided into five subphases, arranged serially as follows:

* *advance green* (GA) – used only with certain staggered start tactics, e.g., to hold a stream in green until a coordinated stream becomes green.
* *fixed time green (GF)* – a green time of fixed duration. For actuated control, this sub-phase is long enough to let the discharge flow stabilize (say, 6 s). For fixed time control, it is the entire green phase.
* *waiting green (GW)* – the sub-phase in which a stream waits if there is no request on a conflicting stream (this sub-phase is skipped for some tactics such as wait-in-red).
* *extension green (GX)* – green time granted on the basis of extension requests; it ends when there is no more extension request, or after a specified maximum duration.
* *parallel green (GP)* – additional green time granted because no other stream can take advantage of the subject stream's green ending, usually because a parallel stream is still in one of the first four green sub-phases, preventing waiting streams from becoming green.

When a stream becomes *active*, it seizes control of the program, forcing conflicting streams to deactivate, allowing the subject stream to become green, and preventing conflicting streams from being activated. The active period includes first four green sub-phases and a short red period preceding the green start during which deactivated streams have their yellow and clearance times.

Model of Cyclic Stream-Based Sequence Control

Sequence Structure

With cyclic sequencing, the traffic engineer specifies a desired *sequence structure* showing the sequence relationships between conflicting traffic streams. Normally, each stream appears once each cycle. An exception is *free realization*, in which a tram, bus lane, or other priority stream with infrequent requests and a small need for green time can be inserted anywhere in the cycle (though not until conflicting streams have become red). Free realization streams do not appear in the sequence structure. It is also possible to have selected streams appear more than once per cycle at specified locations in the control structure, although this option is not discussed further for lack of space.

An intersection can have a large number of possible sequence structures; some are better than others in terms of critical cycle length, lost time, efficient use of extra green time, providing good coordination where desired, and so forth. Several authors (e.g., *11, 12, 13*) have explored the subject of finding the best sequence structure for an intersection under fixed time control with a time-independent arrival process. Many of the same principles apply for actuated control as well, although Bell has rightly observed that the optimal sequence when traffic patterns are assumed known may not be optimal when they are random *(14)*. This paper takes the sequence structure as given. Cyclic sequencing be contrasted with one a-cyclic method used in the Netherlands, in which streams are in turn based on earliest request. While there are some advantages to first in – first out logic when demand is light, its drawback is that when demand becomes heavy, the operation inevitably becomes cyclic, and the signal program may then be locked into an inefficient sequence for a long period of time.

A desired sequence structure can be visualized by means of a *structure diagram* in which streams are arranged vertically in the desired sequence (downward with time). It repeats in principle without end, although for practical purposes, a little less than two full cycles is usually adequate to illustrate all of the stream relationships. A rectangle represents a stream's active period, and arcs show sequence relationships. *Conflict arcs* connect the end of one stream's activation with the next activation of a conflicting stream. *Coordination arcs*, represented by double lines, connect the activation start of simultaneous start streams. Staggered start constraints are also represented by coordination arcs, which connect the later stream's activation start to some point during the former stream's activation. To reduce clutter, the streams that lie in a column form a *conflict group*, meaning that they are all in conflict with each other, so that conflict arcs between streams in the same column are implicit.

Example Intersection

To illustrate these points, an intersection is sketched in Figure 1. Streams 35 and 36 are pedestrian streams; stream 46 is a tram/bus stream in its own right of way; the remainder are general vehicular streams (numbered according to the standard codes common used in the Netherlands)

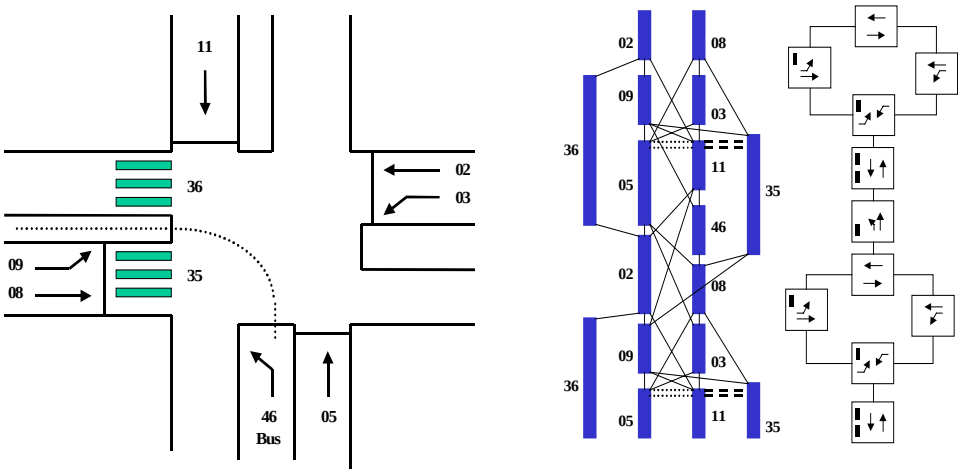

Figure 1. Example intersection Sequence structure Flow diagram

A *sequence structure* and corresponding *flow diagram* (stage flowchart) that will be used in later examples is also given. Some optional tactics for this layout, not incorporated, include: *conditional free realization* for tram stream 46; *truncating other streams' green* upon a request from stream 46; simultaneous start for opposing streams 5 and 11; having pedestrian stream 36 precede stream 11 by, say, 2 s; and providing *green waves* through pedestrian streams 35 and 36, with the direction of the wave dependent on which pedestrian pushbutton was actuated.

Sequence Control and Sequence State

The problem of *sequence control* is to ensure that the signal program advances in a manner consistent with the specified sequence structure and priority tactics. When there is a fixed sequence of stages with no priority interruptions (as in fixed time

control, or in single ring actuated control) this is a trivial matter. But with stream green times varying independently, streams being skipped for lack of demand, and priority interruptions, maintaining the specified sequence without sacrificing the inherent flexibility of stream-based control demands an adequate model of sequence state and sequence change.

The *sequence-state* at any point in time can be illustrated as a cut through the structure diagram. A *feasible cut* separates the structure diagram into an early part (above the cut) and a late part (below the cut) in such a manner that no coordination-arc is cut, and no conflict-arc intersected by the cut goes from the late part of the diagram to the early part. Sequence change can be seen as a matter of advancing the cut through the structure diagram.

Sequence control could be guided by identifying the current cut and referencing it to the structure diagram. Instead, by arranging the structure diagram cyclically, a self-contained way of expressing both the sequence state and the sequence structure is by an array $\boldsymbol{F}$ called the *sequence array* in which element (i,j) is a successor function, defined for conflicting and staggered start stream pairs relative to the current sequence state (cut) as

F_{ij} = 1 if stream *i* follows stream *j*; otherwise F_{ij} = 0

Of course, $F_{ji} = 1 - F_{ij}$.

A feasible sequence array will have at least one stream in *top position,* i.e., with no predecessor (stream *i* is in top position if $\Sigma_j F_{ij}=0$). Likewise, it will have at least one stream in *bottom position,* i.e., with no successor (stream *i* is in bottom position if $\Pi_j F_{ij}=1$). In the TRAFCOD model, active streams are always in top position (although top position streams are not necessarily active).

Successor functions do not apply to stream pairs that are not conflicting, unless they have a staggered start relationship. They also do not apply to stream pairs if either stream has free realization. Sets, sums, and products involving successor functions should always be interpreted as restricted to stream pairs for which a successor function is defined.

Two sequence arrays for the example intersection and sequence structure are shown in Fig. 2. In the first array streams 2 and 8 are in top position; in the rotated array streams 8 and 9 are in top position.

While successor functions have been used in methods for finding optimal sequence structure (e.g., *10, 11, 12*), they have not to our knowledge been used before in a model of sequence control. Of course, many possible values of the sequence array are inconsistent with the sequence structure, and some spell deadlock (e.g., by specifying that stream A precedes B, B precedes C, and C precedes A). It is therefore critical that as the sequence array changes, it remain consistent with the sequence structure.

Simple Rotation

The basic principle of sequence control in a cyclic, demand-responsive operation without a clear cycle start is that *no stream should be served twice until conflicting*

steams have had a turn. This principle is implemented by rotating streams from top to bottom position when their activation ends; this change is called a simple rotation. If stream *i* is in top position (its F_{ij}'s are all 0), a simple rotation is effected by setting all its F_{ij}'s to 1. Graphically, a simple rotation is a shift of the cut representing the current state from passing through (or just above) a stream's active period to just below that active period. A numeric illustration is in Fig. 2, where the second sequence array is a simple rotation from the first (stream 2 was rotated to the bottom).

	2	3	5	8	9	11	35	36	46
2			0		0	0		0	
3			0	1		0			0
5	1	1		1	1				
8		0	0			0	0		0
9	1		0			0	0		0
11	1	1		1	1				0
35				1	1				
36	1								
46		1		1	1	1			

	2	3	5	8	9	11	35	36	46
2			1		1	1		1	
3			0	1		0			0
5	0	1		1	1				
8		0	0			0	0		0
9	0		0			0	0		0
11	0	1		1	1				0
35				1	1				
36	0								
46		1		1	1	1			

Figure 2. Initial stage and simple rotation after end activation of stream 2

It can easily be verified that if a value ***F*** of the sequence array is consistent with the sequence structure, a simple rotation of ***F*** will be as well.

Skipping Streams

When there is continuous demand on every stream and there are no priority interruptions, stream sequence moves smoothly by simple rotation. When one stream rotates to bottom position, any stream that then finds itself in top position becomes in turn, and is activated as soon as possible. However, demand-responsive control requires the ability to activate a stream that is not already in top position, i.e., not next in turn, e.g., because the streams ahead of it have no request, or because the subject stream has a priority request. To ensure that the sequence array remains consistent with the sequence structure, TRAFCOD effects changes in the sequence state by repeated simple rotation. When a stream not already in top position is activated, all the streams above it are rotated to the bottom, allowing the activated stream rise to the top. The streams rotated to the bottom are skipped.

Skipping no-demand streams sometimes allows control programs to uncouple into rings that cycle independently. In the example intersection, suppose that for a period of time the only streams with demand are 2, 3, 8, and 9. They can be divided into two sets, {2,9} and {3,8}, without inter-set conflict or coordination constraint. Therefore these two sets should be able to cycle independently. It is precisely this kind of uncoupling that makes it impossible to identify a unique "cycle start" moment upon which to base sequence control.

Stream skipping is also used in TRAFCOD to allow a fast-moving ring to have a second cycle while a slow-moving ring is still having its first. Streams in the faster ring may then be served twice before a stream in the slower ring has had a turn**.** This exception to the general cyclic sequence control principle makes the control more responsive to variations in traffic.

TRAFCOD FORMULAS

Language Details

TRAFCOD has two main types of variables. *State variables* are either true (1 or non-zero) or false (0). *Timers* take on a user-specified value when initialized, and run down to zero with the clock.

Variable names are letters followed by numeric indices. If there is more than one index, the first refers to the stream. Timers begin with the letter T. The letter N denotes the complement. The prefixes S (start) and E (end) are reserved also. For example, if a variable A becomes true, SA becomes true for one calculation cycle, and then is reset (becomes false). When a variable A becomes false, EA is set (becomes true), and is reset after one calculation cycle.

TRAFCOD expressions use two operators: **and** (**.**) and **inclusive or** (+), with the former having precedence (unless overridden with parentheses). Formulas use either the **assignment operator** (**=**), in which the variable on the left is given the value of the expression on the right, and the **set operator** (**.=**), in which the variable on the left is set if the expression on the right is true. Note: in a statement with the set operator, if the expression on the right becomes false, no change is made to the variable on the left. The formula AN .= *expression* means that the variable A should be reset if *expression* is true.

Formulas are all evaluated in each calculation cycle. Execution sequence among formulas is arbitrary except where specified. With every time step (the default increment is 0.1 s), detections are noted first, and a calculation cycle is executed. If any variable changes value during the calculation cycle, another calculation cycle is executed without advancing the clock. A warning is issued if variables have not stabilized after a reasonable number of calculation cycles.

The main TRAFCOD variables and formulas are summarized in Table 1; most of them are also explained in the following sections. For lack of space, some tactics are not included in this set of formulas. The index *i* denotes the subject stream. Other indices have the following meaning:

d	detector index for stream *i*
f	free realization stream in conflict with i
k	stream in conflict with i, not including free realization streams
kf	stream in conflict with *i*, including free realization streams
kw	stream in conflict with *i* that will not yield to a fill-in request
p	stream parallel (not in conflict) with stream *i*
ps	stream with a simultaneous start relationship with *i*
psc	stream with a simultaneous start common request relationship with *i*

Requests for green

Requests for green (*Q)*come primarily from detectors. The formula for a ***detection-based request*** (*QD*) depends on the detector configuration and desired detection strategy; for example, two possible formulas are

$$QD_i .= R_i . \Sigma_d (D_{id})$$ (any detector can trigger a request during red) (1a)

$$QD_i .= R_i$$ (traffic assumed) (1b)

Reasons for using the "traffic assumed" strategy include that a stream has no detectors, or is favored. The reset formula for *QD* also depends on detection strategy; the default formula is

$$QDN_i .= GP_i.XN_i \quad (2)$$

This formula resets QD before the end of the green phase, unless the green phase ends without fulfilling the extension green request(X).

Extension green requests(X) are based on detections. With the standard gap-seeking strategy, detections initialize a corresponding gap timer (whose user-specified initial value depends on factors such as loop length and position) which requests an extension until it runs down to zero:

$$X_i = \Sigma_d TDH_{id} \quad (3)$$

Other extension green request strategies can be programmed as well. For example, under a strategy to extend green for buses detected several seconds before the stop-line, a bus detection could increment a counter, which is decremented as buses pass an exit detector. A nonzero value of the counter could then be a basis for an extension request. If information on expected arrival patterns and queue lengths on conflicting streams is available, it can be used along with more complex logic to determine whether to continue an extension request.

Requests for green can also be generated using an optional tactic called **fill-in request** (QF), which generates a forced request for a stream without traffic if it is in turn and its green won't interfere with another stream. Conflicting green (G_{kw})has priority over a fill-in request, unless specified otherwise (e.g., for a low volume stream not expected to make effective use of parallel green). If there are conflicting fill-in requests (QF_k), priority goes to the one that comes earliest in the sequence:

$$QF_i = Ri.ONi. \Pi GN_{kw}.QN_i.YN_i.YMN_i. \Pi(QFN_k+F_{ki}) \quad (4)$$

To ensure that priority among conflicting fill-in requests is based not on which is processed earliest in the calculation cycle but on which is next in turn, the variable QF is not acted on in equation 5 unless it is still true after a complete calculation cycle.

Also a **simultaneous start with common request** (Q_{psc}) tactic can generate requests for green.

Requests can be temporarily overridden by the omit function (O) which shuts down a stream temporarily, e.g., because a drawbridge is up or for temporary construction. Thus, the formula for a general request on stream i is

$$Q_i = ON_i.QD_i+R_i.ON_i. \Sigma Q_{psc}.R_{psc}+QF_i.SQFN_i \quad (5)$$

The unusual notation associated with the fill-in request QF_i. $SQFN_i$ means QF_i is true, but has not just started.

Activation

The activation formula is:

$$A_i .= (Q_i+QA_i).YN_i.YMN_i.R_i \quad (6)$$

A stream with a request will be activated unless it must yield (state variable *Yi*) to a stream with higher priority:

$$Y_i = \Sigma A_k.Q_k + \Sigma[F_{ik}.(Q_k+QA_k)] + \Sigma Q_f Z_i \quad (7)$$

Simultaneous start streams must be activated together, and therefore must yield mutual to each other(*YM*):

$$YM_i = \Sigma Y_{ps} \quad (8)$$

Another way to activate a stream is the *activation-only requests(QA).* It is used only for simultaneous start streams that have no request when a partner stream does (while the no-demand stream may not become green, it must still be activated with its partner in order to preserve the sequence structure):

$$QA_i .= AN_i.\Sigma(Q_{ps}.AN_{ps}) \quad (9)$$

$$QAN_i .= A_i \quad (10)$$

A stream activated on the basis of an activation-only request is skipped (*UC*, skip for coordination) as soon a simultaneous start partner becomes green:

$$UC_i = A_i.R_i.QN_i.\ \Sigma SG_s \quad (11)$$

Activation ends when a stream enters parallel green, or is skipped before becoming green:

$$AN_i .= SGP_i + R_i.(O_i+Y_i+UC_i) \quad (12)$$

Green start

The green start formula is:

$$G_i .= Ri\ .A_i.Q_i.BN_i.BMN_i \quad (13)$$

Just as *Y* and *YM* prevent a stream from being activated, stronger conditions, *B* (block) and *BM* (mutual block) prevents an activated stream from becoming green. The main reason for a block is a conflict (*K*, whose formula is not given here*)*, which lasts from start green of any conflicting stream until all conflicting streams are red and their specified clearance times have elapsed:

$$B_i = K_{i+}Y_i+TRGN_i \quad (14)$$

$$BM_i = \Sigma B_s.A_s.Q_s \quad (15)$$

Continuing Green

Normally, after a stream becomes green, it rests in waiting green until there is a request on a conflicting stream. That way, when a request from a conflicting stream finally comes, the stream that is green has the chance to go into extension green and finish serving any existing queue before yielding its green. Alternatively, waiting green can be skipped in keeping with the wait-in-red tactic (i.e., all signals wait in red when there is no demand so that an arriving vehicle on any stream can be given an immediate green; this tactic is standard in Sweden *(5,7)*):

$$CW_i = \Sigma(QN_{kf} + QAN_{kf}) \quad \text{(default) (16a)}$$

$$CW_i = 0 \quad \text{(wait-in-red) (16b)}$$

Parallel green is the time after a stream has ended its activation and rotated to the bottom position, yet remains green because no conflicting stream could become green if the subject stream became red. With the wait-in-red strategy, parallel green should only continue as long as a parallel stream is active. Another alternative, a variation of wait-in-red, is to not allow any parallel green at all as a safety measure, effectively permitting cars to enter the intersection only from a queue or in a platoon connected to the initial queue:

$$CP_i = \Pi(AN_{kf}+QN_{kf}) \quad \text{(default) (17a)}$$

$$CP_i = \Pi(AN_{kf}+QN_{kf}).(\Sigma A_p) \quad \text{(wait-in-red) (17b)}$$

$$CP_i = 0 \quad \text{(force platooning) (17c)}$$

Sequence Control Formulas

As mentioned earlier, TRAFCOD allows the sequence variables F_{ij} to change only by simple rotation, in which a stream in top position moves to bottom position. Rotation is activated *(AR)* when a stream's active period ends, or when it is in the top position and is supposed to be skipped. The general skip condition *U*, used to bring an activated stream to top position, applies to every stream immediately preceding an activated stream and to every stream preceding a stream with a skip condition. The rotation and skip formulas are

$$U_i .= \Sigma F_{ki}.(A_k+U_k) \quad (18)$$

$$AR_i = (\Pi F_{ki}).(GP_i+U_i+UC_i) \quad (19)$$

$$F_{ik} .= AR_i \quad \textit{for all} \text{ k } \textit{conflicting with} \text{ i} \quad (20)$$

$$FN_{ki} .= F_{ik} \quad \textit{for all} \text{ k } \textit{conflicting with} \text{ i} \quad (21)$$

$$UN_i .= AR_i \quad (22)$$

The calculation sequence must ensure that the AR_i formula, the F_{ik} formulas (for all k conflicting with i), and the UN_i formula are executed in that order. Skipping and rotation formulas do not apply to free realization streams.

Sequence Control Examples

Three examples based on the intersection and sequence structure of Fig. 1 will illustrate sequence control using these formulas. First, suppose streams 2 and 8 are just finishing their fixed time green. Streams 9 and 11 have requests, but are yielding to active and/or prior streams. Because of stream 11's request, streams 2 and 8 skip through waiting green into extension green. When stream 2's extension condition ends, it goes to parallel green, deactivates, and rotates to bottom position, leaving stream 9 in top position. Stream 9's yield condition ends, and it is activated, ending stream 2's parallel green, which was momentary. Stream 9's green start is blocked until stream 2's yellow phase ends and the clearance time between streams 2 and 9 has elapsed, whereupon stream 9 becomes green.

Suppose stream 8 ends its extension green before stream 3. At that moment, stream 8 enters parallel green, ends its activation, and rotates to bottom position. Because stream 11 cannot be activated yet (it is still yielding to stream 9), stream 8 remains in parallel green. When stream 9 ends its extension green, it too enters parallel green, is deactivated, and rotates to the bottom. This ends stream 11's yield condition, allowing stream 11 to be activated, forcing stream 8 and 9 to end their parallel green. At this point stream 3, which has no request, is still in position above stream 11, so it gets a skip condition. Because it is in top position, it is immediately rotated to the bottom, leaving streams 5, 11, 35, and 36 in top position. Stream 11 soon becomes green. Whether streams 5, 35, and 36 also become green depends on the chosen tactic for fill-in green.

A second example uncoupling into rings that cycle independently. Suppose is no demand for a period of time on any stream except 2, 3, 8, and 9. These streams form two rings {2,9} and {3,8} with no inter-ring conflict. Suppose streams 2 and 3 are green and active, in top position, and there are requests on streams 8 and 9. If stream 3 finishes service before stream 2, it is rotated to the bottom, leaving stream 2 alone in top position. Stream 8, which no longer has to yield, is activated, triggering a cascade of skips: streams 11, 5, and 46 get a skip condition because they precede stream 8, and streams 9 and 2 get a skip condition because they precede streams with a skip condition. The first stream to be rotated to the bottom is then stream 2 (which is currently active and green), followed by stream 9, then stream 11 and 5, and finally stream 46, leaving streams 2 and 8 in top position. Relative to the sequence structure, stream 8's activation advanced the sequence state almost a full cycle, skipping an active stream (2) momentarily before returning it to top position.

A third example, based again on the same sequence structure, illustrates a faster-moving ring double cycling within a slower moving ring. Suppose streams 8 and 9 are in top position, active and green, with pending requests on streams 3, 46, and 2. Streams 2 and 46 are in bottom position. If the first stream to deactivate and rotate to the bottom is stream 9, stream 2 will be activated, skipping streams 5 and 11 and essentially starting a new cycle for ring 2-9-5. However, the skip conditions placed on streams 5 and 11 will in turn trigger skip conditions on *their* predecessors - streams

3 and stream 8. Stream 8, which is active rotates first to the bottom position, followed by stream 3, then streams 5 and 11. At this point stream 2 has reached top position. However, stream 8, which is active, has not yet returned to top position, so a skip condition is issued to its predecessor, stream 46, which rotates to the bottom, leaving stream 8 back on top. The effect of these rotations was to enable ring 2-9 to start a new cycle while ring 9-46 was skipped through, because ring 8-3-46 was moving so slowly.

Other Control Tactics

Space limitations prevent a complete description of how other control tactics can be modeled in TRAFCOD. A brief description for some tactics follows.

Priority truncation. A truncate condition *(Z)* is applied based on desired priority tactics. For example, to truncate conflicting streams in response to priority requests (externally defined),

$$Z_i = \Sigma\,[\text{priority request on stream k}] \qquad (23)$$

Conditional free realization means a stream has free realization for priority requests, but otherwise must wait its normal turn. This tactic is used for improving punctuality on a public transport line by giving it priority only when it is late relative either to a scheduled time or to a headway *(15)*. That way, early buses or trams will be "held" until their turn in the signal cycle, while late vehicles pass through without delay. This tactic is modeled by superimposing a free realization stream on top of a normally sequenced stream.

A variety of *staggered start tactics* can be modeled, depending on the size of the lag and the desired push-pull relationship between the streams.

Application Software

Two sets of application software have been developed for testing TRAFCOD: a controller simulator, and a formula generator. The former, written in C, simulates the intersection controller by evaluating, at each time step, the given formulas. Signal indications of the various streams are displayed graphically. By hitting keys, the user activates and deactivates presence detectors, which drive the simulation and enable the user to test its operation. This program includes valuable debug facilities, including displaying state variables and tracing all state changes, by which the user may check that the controller is operating as expected.

The formula generator, also written in C, generates site-specific formulas that can feed a controller (or the controller simulator). Its only inputs are:

1. a conflict matrix, generalized so that each cell expresses the relationship between a pair of streams (e.g., conflict, simultaneous start, etc.)
2. a matrix indicating the sequence structure
3. a list of stream attributes, including whether a stream has free or conditional free realization, power to truncate conflicting streams, and fill-in requests, whether it will yield to fill-in requests, and the preferred waiting tactic (in green or in red, with or without forcing a platoon).

Results

The controller simulator has been used for several years as a training tool at the Delft University of Technology, and is considered to be very reliable. The formula generator has been tested with the controller simulator on a variety of intersections using the full range of control options described in this paper (i.e., free realization, truncation, wait-in-red, fill-in traffic). The formulas work as expected, controlling the signals in keeping with the selected tactics without any deadlock or unexpected side effects. Formula generation has been reduced from a matter of days (using manually generated formulas and ad hoc rules that must be tested to see if they result in the desired manner of control) to less than an hour to prepare the inputs needed for the formula generation program. We hope to be able to field test the new sequence control method with its automated formulas soon in the Netherlands.

The potential benefits of a sequence model that permits general formulas for flexible stream-based control is large. For traffic engineers currently using stream-based control, it frees them from spending a lot of effort to ensure that their control formulas have been written correctly written so that they can instead give attention to higher level decisions such as which control tactics to use. It also frees their control programs from artificial restrictions used to make up for the deficiencies of simpler sequence models. For traffic engineers considering a switch to stream-based control, automatic formula generation makes stream-based control much more accessible, putting within reach a powerful method for traffic signal control that is exceptionally responsive, is well adapted to complex intersections, and is easily programmed to handle a wide variety of priority and other control tactics.

References

1. MCE 0141: Microprocessor based traffic signal controller for isolated linked and urban traffic control installations. Department of Transport (UK), 1984.

2. Hallworth, M.S. 'High-performance' signal controllers. *Traffic Engineering & Control 21* , pp. 242-248, 1980.

3. Heydecker, B.G. and I.W. Dudgeon. Calculation of signal setting to minimise delay at a junction. In *Transportation and Traffic Theory*, (N.H.Gartner and N.H.M. Wilson, ed.). New York: Elsevier, pp. 159-178, 1987.

4. van Zuylen, H. Acyclic traffic controllers. Note from the Transport Studies Group, University College London, 1976.

5. Peterson, A., T. Bergh, and K. Steen. LHOVRA – a new traffic signal control strategy for isolated junctions. *Traffic Engineering & Control 27*, pp. 388-389, 1986.

6. Guidelines for traffic signal installations (in German). Forschungsgesellschaft für Strassen- und Verkehrswesen, 1981.

7. Kronborg, P. and F. Davidsson. MOVA and LHOVRA: Traffic signal control for isolated intersections. *Traffic Engineering & Control 34*, pp. 195-200, 1993.

8. Ward, D.E. and J.R. Landles. The American double ring system applied in London. Proc. 6th International Conf. on Road Traffic Monitoring and Control, Institution of Electical Engineers, London, 1992.

9. Hakkesteegt, P. TRAFCOD: Method for systematic traffic control design. Delft University of Technology, Transportation Research Laboratory, Report VK4200.202, 1988.

10. van Grinsven, T. Software toolkit for the design of traffic signal rules in the C programming language: CCOL/C-REGELAAR, version 5.0 (in Dutch). Van Grinsven Software, 1998.

11. Cantarella, G.E. and G. Improta. Capacity factor or cycle time optimization for signalized junctions: a graph theory approach. *Transportation Research 22B(1),* pp. 1-23, 1988.

12. Gallivan, S. and B.G. Heydecker. Optimising the control performance of traffic signals at a single junction. *Transportation Research 22B(5),* pp. 357-370, 1988.

13. Wong, S.C., W.H.Law, and C.O.Tong. Determination of optimal successor function in phase-based control using neural network. Proceedings of the IEEE Intelligent Vehicle Symposium, pp. 120-125, 1996.

14. Bell, M.G.H. A probabilistic approach to the optimization of traffic signal settings in discrete time. In *Transportation and Traffic Theory,* M. Koshi (ed.), Elsevier, 1990, pp. 619-632.

15. Furth, P.G. and Th.H.J.Muller. New concepts in transit priority at signalized intersections. Presented at the Transportation Research Board Annual Meeting, 1998.

Table 1: TRAFCOD Formulas

Variable	Meaning	Formula
A	**activate**	$A_i .= R_i.(Q_i+QA_i).YN_i.YMN_i$
AR	**activate rotation**	$AR_i = (\Pi F_{ki}).(GP_i+U_i+UC_i)$
B	**block green**	$B_i = Y_i+K_i+TRGN_i$
BM	**mutual block**	$BM_i = \Sigma B_s.A_s.Q_s$
CA	**continue advance green**	***formula depends on staggered start tactic***
CP	**continue parallel green**	***see equations 17a, 17b, 17c***
CW	**continue waiting green**	***see equations 16a, 16b***
D_{id}	**detection on detector *d* of stream**	***depends on actual traffic***
F	**follow**	$F_{ik} .= AR_i$
		$FN_{ki} .= F_{ik}$
G	**green**	$G_i .= R_i.A_i.Q_i.BN_i.BMN_i;$
		$GN_i .= GP_i.TGGN_i.(CPN_i+O_i)$
GA	**advance green**	$GA_i .= SG_i$
		$GAN_i .= SGF_i$
GF	**fixed time green**	$GF_i .= GA_i.(CAN_i+Z_i+O_i);$
		$GFN_i .= TGFN_i+Z_i+O_i$
GP	**parallel green**	$GP_i .= EGX_i$
		$GPN_i .= EG_i$
GW	**waiting green**	$GW_i .= EGF_i$
		$GWN_i .= SGX_i$
GX	**extension green**	$GX_i.= GW_i.(CWN_i+Z_i+O_i)$
		$GXN_i.= XN_i+TGXN_i+Z_i+O_i$
K	**conflict**	***formula internal to conflict module (not accessible to user)***
L	**yellow**	$L_i .= EG_i$
		$LN_i .= TLN_i$
O	**omit**	***externally set***
Q	**request**	$Q_i = ON_i.QD_i+R_i.ON_i.\Sigma Q_{psc}+QF_i.SQFN_i$
QA	**activation request**	$QA_i.= AN_i.\Sigma(Q_s.AN_s)$
		$QAN_i.= A_i$
QD	**detector request**	***user defined; see equations 1a, 1b, and 2***
QF	**fill-in request**	$QFi=R_i.ON_i.\Pi GN_{kw}.QN_i.YN_i.YMN_i.\Pi(QFN_k+F_{ki})$
R	**red**	$R_i .= EL_i$
		$RN_i .= SG_i$

TDH	**detector timer**	***initialized with detection***
TGF	**fixed time green timer**	***initialized at start of fixed time green***
TGG	**green guarantee timer**	***initialized at start of green***
TGX	**green extension timer**	***initialized at start of extension green***
TRG	**red guarantee timer**	***initialized at start of red***
U	**skip**	$U_i .= \Sigma F_{ki}.(A_k+U_k)$ $UN_i .= AR_i$
UC	**coordination skip**	$UC_i = A_i.R_i.QN_i.\Sigma SG_s$
X	**extention request**	***depends on tactic; see equation 3***
Y	**yield**	$Y_i = \Sigma A_k.Q_k + \Sigma F_{ik}.(Q_k+QA_k)+\Sigma Q_u+Z_i$
YM	**mutual yield**	$YM_i = \Sigma Y_s$
Z	**truncate**	***depends on tactic; see equation 23***