Project
Design and Organization of Autonomous Systems :
Intelligent Traffic Light Control

Emil Nijhuis, 9917527
Stefan Peelen, 0010065
Roelant Schouten, 0010774
Merlijn Steingröver, 0043826

Supervisor: Bram Bakker

3rd February 2005

Abstract

Because of the increasing density of the traffic flow in urban areas there is a need for optimal performance of traffic lights. In this paper we will describe an existing approach of reinforcement learning applied to the optimization of traffic light configurations, and introduce a new approach. Our approach uses implicit cooperation between traffic lights, letting cars take into account the traffic situation of the road ahead. We will show that this approach outperforms the existing algorithm, using experiments performed with a traffic simulator.

1 Introduction

In our current world traffic has become an increasingly substantial part of society. With people becoming ever more mobile, traffic jams are becoming a more and more common sight, especially in large urban areas. This, of course, leads to all kinds of unwanted side effects like people arriving too late at their destination, economical damage and environmental pollution.
Traffic in urban areas is mainly regularized by means of traffic lights, which can make for unnecessary long waiting times for cars when not efficiently configured. This inefficient configuration is unfortunately still the case in a lot of urban areas; most of the traffic lights are based on a 'fixed cycle' protocol, which basically means that the lights will be turned on green for a fixed amount of time and consecutively on red for a fixed amount of time.
This, of course, is not optimal, since such a policy does not take into account the current traffic situation at a particular traffic light, meaning that situations can occur in which a long queue of waiting cars is being held up by a shorter queue (or in the worst case, by no queue at all) because of the traffic light's policy. What you would want to happen in such a situation is that the policy decides to let longer queues have an 'advantage' over the shorter ones in order to increase the total traffic flow. This of course should not lead to situation where cars at a quieter part of a junction will never be able to continue their journey.

The question that arises is how to exactly define this policy that does take into account different traffic situations at traffic light junctions. It is likely that trying to hard-code all the possible situations concerning a traffic junction, and the traffic light configurations according to those situations, in some fashion will require an unreasonable amount of time and effort, and possibly such an approach could lead to incomplete representations.
A more feasible approach would be to make use of techniques existing in the field of Artificial Intelligence. Such methods often have the ability to cope with large representations such as those appearing in our traffic light problem by in some way automatically learning the correct policy. Different A.I. based techniques have already

been tried in attempt to solve the traffic light problem; examples are genetic algorithms, fuzzy logic and reinforcement learning.
It is on this last mentioned technique that the emphasis of this report lays. We use reinforcement learning to learn for each traffic light junction in a city the optimal configuration of the lights.
How this works exactly will be shown in the following sections, starting with an introduction to reinforcement learning as well as a description of the traffic simulator we used for our experiments in section two. In section three we will introduce our approach to reinforcement learning, followed by experiments performed with this approach in section 4. The paper will be rounded up by the conclusions and discussion in section 5.

2 Reinforcement Learning, GLD, MAS Learning

There are two common approaches of modelling traffic: macroscopic and microscopic models. In our paper we focus on traffic in the city environment, hence we will only consider only microscopic models. The provided traffic simulator *Green Light District Simulator* models traffic as Multi-Agent System [1], where the agents are the traffic lights in the infrastructure.
There are several methods for learning a policy for these agents, such as genetic algorithms and reinforcement learning. As stated above, the technique we are using in order to learn an optimal traffic light configuration is reinforcement learning. This method has already been used in different domains with considerable success. Arguably the most impressive result so far is in backgammon [6], where a program based on this method is able to play at human expert level.
In this section we will first give an overview of this technique, to give the reader a general background of how it works; successively we will show how reinforcement learning and Q-learning is used in the traffic environment.
In the last subsection we give a description of the traffic simulator we used to test the theory.

2.1 Reinforcement Learning

Reinforcement learning is a learning technique for learning control strategies for autonomous agents from trial and error, [3], [4], [5]. The agents interact with the environment by trying out actions and use resulting feedback (reward, and the consecutive state) to reinforce the behavior that leads to a desired outcome (see figure 1 for illustration).

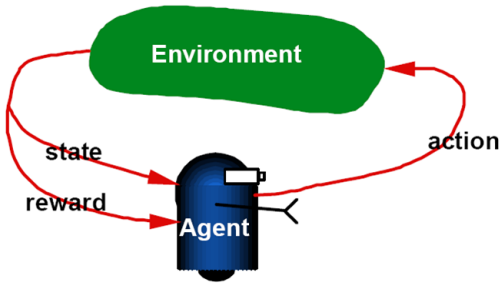

Figure 1: An agent and his world

Considering our environment that means we want to optimize the total waiting times of all the cars in the infrastructure. That means our algorithm should reinforce traffic light behavior that will minimize the waiting time of all the present and future cars in our infrastructure.

For this we first need a formal structure of the model. The traffic optimization problem can be modelled as a Markov Decision Process (MDP) which makes reinforcement learning a suitable approach for our task. This consists of a finite set of states $S = s_1, s_2, ...$, a finite set of actions $A = a_1, a_2, ...$ an agent can perform, the probabilities $P(s, a, s')$ which are the probabilities that an agent in state s will arrive in state s' after performing the action a and finally a real valued reward function $R : S \text{ x } A \rightarrow \mathbb{R}$ defines a reward which an agent will receive when arriving in state s after executing a. The task of our agents is then to determine a policy $\pi : S \rightarrow A$, which selects an action a_t for a state s_t at time t that optimizes the future discounted reward:

$$V^{\pi}(s_t) \equiv r_t + \gamma r_{t+1} + \gamma r_{t+2} + \gamma r_{t+3} + ... \equiv \sum_i \gamma^i r_{t+i}$$

where γ is the discount factor and r_t are the rewards at time t. The most common way to do this algorithmically is using Q-learning. We define an evaluation function $Q(s, a) : S \text{ x } A \rightarrow \mathbb{R}$ where the value represents the maximum discounted future reward. Our optimal policy for our agents will then be: $\pi^*(s) = argmax_a Q(s, a)$. An algorithm to obtain the right Q-values is given in [3].

2.2 RL in Traffic Light Controllers

Now that we have established a basic notion of reinforcement learning, it's time to show how exactly this can be

applied to our earlier described traffic problem.

First we will introduce the main entities with which we will reason in the algorithms: the cars driving through the city and the traffic light junctions controlling the individual traffic lights and thus the traffic flow through the city. Furthermore there are the roads on which the cars drive, which consist of different driving lanes going to different directions at a junction.

The goal of traffic light control in a city is to minimize waiting times in front of traffic lights and to minimize the total travel time. The obvious way to solve this problem would be to optimize the configurations of the traffic light. In addition to this approach we also might want to change the behavior of the car driver by telling him how he should travel to his/her destination (this is called co-learning). Co-learning has already been explored in [1] and showed good results in an urban like infrastructure.

In our project we focused mainly on the traffic light controller without manipulating the available driving policies of the cars. From here, two different views can be employed on how a reinforcement learning algorithm could be applied to the domain.

- One is a **traffic light-based** approach, which takes the traffic light junctions to be the learning agents. Such an approach would require each traffic light junction to learn a value function that maps the waiting times of all the cars at each of it's drive lanes to a traffic light configuration. This approach would lead to a potentially huge state-space, because of all the different possible configurations of cars on the driving lanes [7].

- The other is a **car-based** approach, which takes each individual car to be an agent. This is more of a multi-agent based approach; the idea is that each car predicts it's waiting time in the case of a green and of a red sign, now all these predictions/estimates can be combined to make a decision for the traffic light. Note how cars don't really have to represent the Q-values themselves; the representation is simply car based

To avoid a large state space with the traffic light-based approach, the software we worked with uses the car-based approach.

The state of a car is described as a quadruple $s = [n, p, d, nr]$ of its current position in terms of the next junction (node) n it approaches, the position in the current driving lane p, its final destination d and its position in the queue in front of the consecutive traffic light nr. We consider then two actions of the consecutive traffic light $a = red, green$. Hence we need to calculate all the Q-values $Q(s, a)$. After that we can derive the optimal traffic light configuration A_j^{opt} for a particular traffic light as follows:

$$A_j^{opt} = max_{A_j} \sum_{i \in A_j} \sum_{s \in localcars} Q(s, red) - Q(s, green)$$

Reaching the next state a reward will be given and the Q-values will be updated with:

$$Q(s, a) = \sum_{s'} [P(s, a, s') * R(s, a, s') + \gamma V(s')]$$

this time the V function will be computed as follows:

$$V(s) = \sum_a P(a|s) * Q(s, a)$$

Further is the reward 1 if the car has to wait or cannot move and is 0 otherwise. The result is that the algorithm will try to optimize the total waiting times of all the present and future cars.

2.3 GLD

We used a traffic simulator called "Green Light District" (GLD). This is a JAVA based application built by the group of Marco Wiering at the Universiteit van Utrecht. GLD allows to create custom 'cities' or traffic networks using a straightforward point-and-click interface. It is possible to connect junctions (or nodes) with roads, which themselves have a certain number of driving lanes, represented as discrete cellular automata. If more than two roads are connected to a junction it will automatically become a traffic light junction, which itself follows some simple rules to make sure it will not cause any accidents by putting too many lights on green.
At the edge of the network there are nodes which let cars enter the network; these so called 'edge nodes' have a certain spawning rate which defines how fast it will let new cars enter the network; the edge nodes are at the same time, the destination nodes for the cars.

The cars are the main agents in this framework; they are spawn at the edge nodes and get another edge node assigned to be their destination. This destination is part of their state-space. Cars have a very limited set of actions and a limited state space; the actions they can perform are driving at a certain speed, waiting in a traffic queue or crossing an intersection.
It also is worth mentioning that each car has a path-planning module available, which calculates the path from their current position to their destination node so they know at each junction which way they have to go.

The simulator is capable of keeping track of several statistics over time, both at node level and at global level.
At the node level it keeps track of the total number of cars that crossed the junction, and also the average time that a car has to wait before that particular junction is recorded. On the global level the system can keep track of the average junction waiting time, the total amount of cars that are generated but cannot enter the city and, most importantly, the average trip waiting time of all cars.

The GLD simulator already incorporates several traffic light controllers; there are options to run the simulator using both learning and non-learning controllers. Learning controllers include reinforcement learning, evolutionary learning or neural network-based algorithms. Non-learning controllers include simpler algorithms that will, for example, set the traffic light with the longest waiting queue to green.

3 Our Implementation

As said above, the simulator already has a reinforcement learning algorithm implemented, which turned out to work very well. Our assignment now was to try and improve the current algorithm in some manner by making a novel variation on the existing implementation.

To accomplish this we had to figure out what the weaknesses of the current implementation were, or where there was room for improvement. We found that the main shortcoming was the lack of cooperation between the traffic light junctions. Although each junction was able to come to an optimal configuration for itself, this configuration could easily turn out to be only locally optimal because the cars, although they can traverse through the current junction, will be getting stuck again at the next one. If some level of cooperation would exist between the traffic junctions, they could take into account the congestion at other junctions and come to a more globally optimal configuration.

The question now was how to incorporate this idea in the current framework. A few options can be thought of, like trying to build a 'meta-controller' which would, in some fashion, learn the optimal global traffic light configuration; we also thought about trying a more game theoretic approach, trying to maybe establish a Nash equilibrium between the individual junctions. However straightforward algorithms for these approaches are not known, hence they were not viable options for our 1 month project.

We chose a more implicit approach; we let the individual cars take into account the traffic situation not only at their current traffic light, but also at the next traffic light they will visit.
We found that an elegant way to implement this was by incorporating the traffic situation of the next traffic light a car will visit into it's state space. What happens here is that each car will have it's state space extended with an extra dimension which represents the traffic situation at the next traffic light. Remember from section 2 that a car's state space is defined by the current node, the direction, the queue position and the destination; this will now be expanded with another element, congestion, which represents whether or not the next traffic node is congested.
This congestion is represented as a binary, a next node is either congested, or it is not. Congestion is calculated as the ratio between the number of cars on a drive lane and the total capacity of that lane. If that ratio passes some fixed threshold, the lane is congested and a car with that lane as it's next target will 'know' this. This knowledge will now be taken into account when the Q values for the next step are calculated, because it is represented in the state. We named this algorithm TC-CBG which stands for "congestion be gone"

Another, somewhat more simple, approach to this same idea is to make use of a heuristic which takes into account the traffic situation ahead. Instead of explicitly representing future congestion in the state space, we can also let the congestion be a factor in the calculation of

the traffic light's decision.
This calculation again looks ahead to the next driving lane the car will visit, and checks whether it is congested or not; only now this congestion rate is not mapped to a binary value, but kept a real value. This decimal is now used in the estimation calculation by multiplying the current estimate with 1 minus the congestion rate. The congestion rate is subtracted from 1 to make sure that the calculated gain will be taken fully into account when the next lane is empty (it is then multiplied by 1), or will not be taken into account at all if the next lane is fully congested (it is then multiplied by 0). This can be formalized as follows:

$$A_j^{opt} = max_{A_j} \sum_{i \in A_j} \sum_{s \in l} [(1-c(s))(Q(s, red) - Q(s, green))],$$

where l sums over all local cars at this traffic light and $c(s)$ is the appropriate congestion factor for each car. We have named this approach HEC, which stands for Heuristic Enhanced Controller.

4 Experiments and Results

We tested our algorithms using the GLD simulator, we used the same traffic network used in the experiments of the original algorithm in [1]. We compared our test results to the results of the original implemented reinforcement learning algorithm (TC-1). We did not use the test results as described in the paper, but ran all tests ourselves, to make sure both approaches were tested under the same circumstances.
Furthermore we ran some tests of our own, incorporating different spawning rates throughout time. We did this because we thought this would make for a more realistic traffic simulation in a city; it is quite unrealistic to think that traffic keeps on coming into a city at the same rate all day long. In reality there are situations like rush-hours or night hours, due to which the incoming traffic flow fluctuates.
We implemented a dynamic spawning rate option, which can change the spawning rate of a particular edge node over time. Results of the experiments performed are given below.

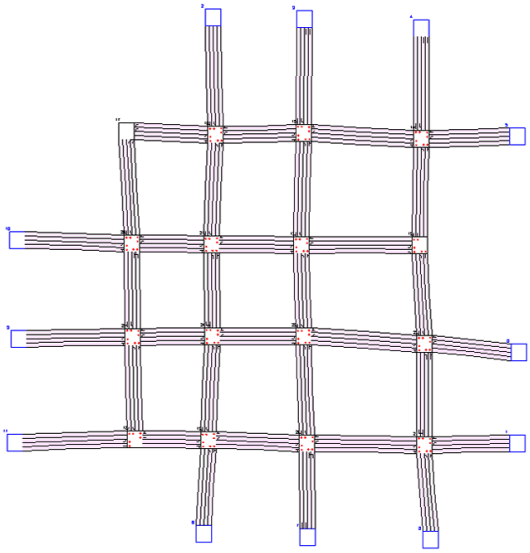

Figure 2:Our test city, with 12 edge nodes, 15 traffic signs and 1 junction that doesn't have a traffic light.

4.1 Simulation 1

This simulation mimics an experiment performed by Wiering in (experiment 1, a large infrastructure). In this simulation all edge nodes have a spawn rate of 0.4, this has been shown to be the maximum possible rate for the original algorithm to still perform [1], if it's set higher, the roads will not be able to cope with the high amount of cars that enter the city. We let the different algorithms run for 50.000 cycles and averaged the result of 5 such simulations.

Each algorithm was run twice, once in it's normal form, and once with the co-learning (CL) feature turned on. This co-learning is a feature implemented in the original software. The idea behind it is that cars can have their path-planning module being influenced by information gained from other cars about the traffic situation on the roads ahead. Using this information a car can choose not to take that road but consult it's path planning module for a different path to it's destination, avoiding busy roads. In previous experiments [1] usually co-learning had a beneficial effect on the results.
From these results it can be seen that our HEC algorithm always outperforms the other two, both with co-learning and without. Interesting to note is that co-learning doesn't seem the have much of an impact on the performance of the CBG-algorithm.

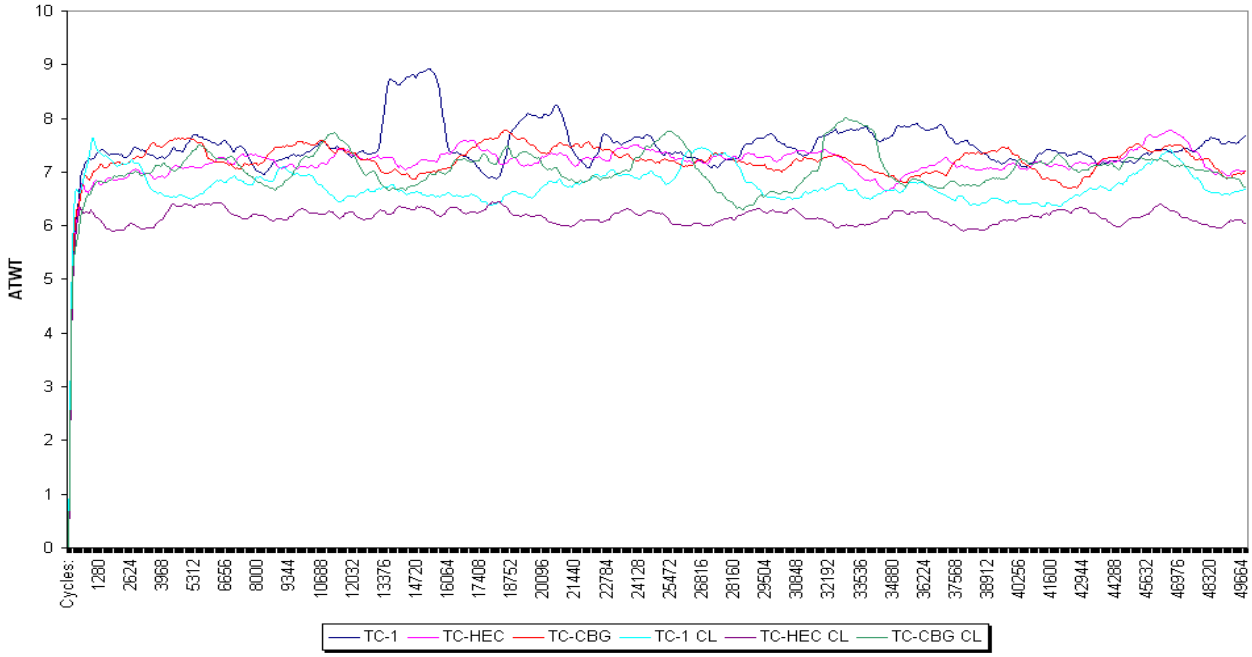

Figure 3: Comparison of several implemented algorithms vs. the existing ones

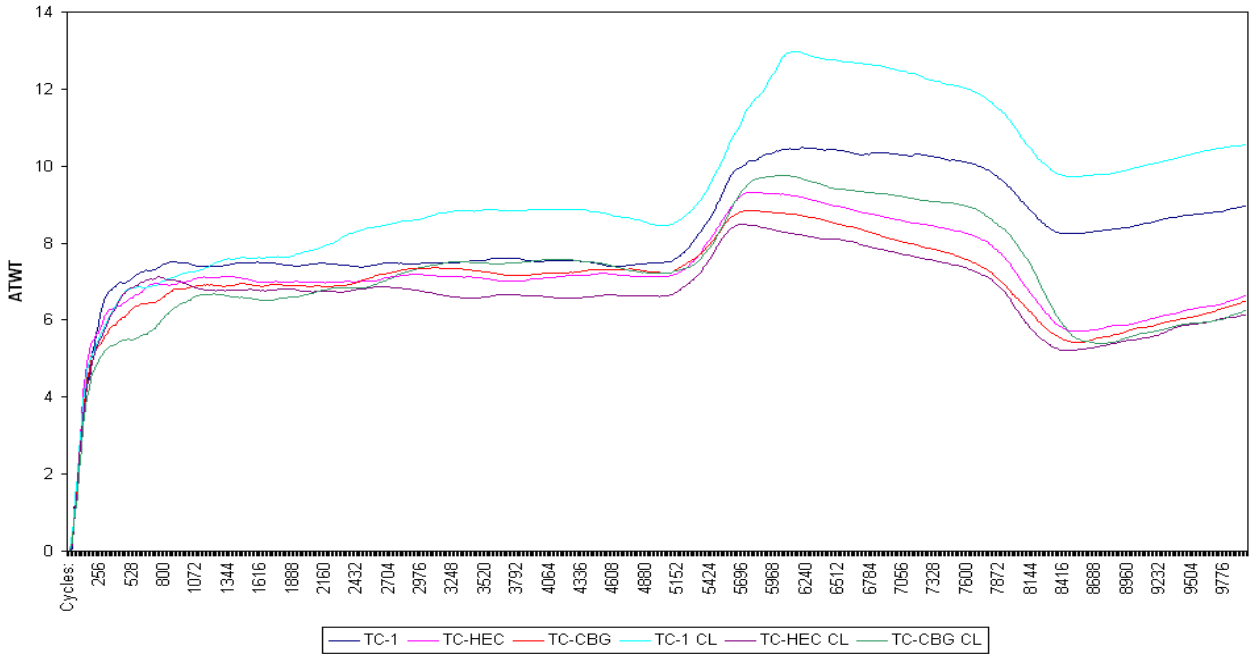

Figure 4: Comparison of several implemented algorithms vs. the existing ones in handling rush-hour

4.2 Simulation 2

This second experiment incorporates the dynamic spawning rate mentioned above. Note that this simulation is not run over 50.000, but over 10.000 cycles. The results are still an average over 5 test runs.
The dynamic spawning rate affected two edge nodes. All edge nodes were initialized at a spawning rate of 0.4, but at cycle 5000, rush hour started and the spawning rates of the two affected nodes rose to 0.7; at cycle 5500 the rate was changed to 0.2 for a 'cool down' period, and at cycle 8000 it was changed again to the original 0.4 and finished the run to the final 10.000 cycles.

The graph shows that in the situation with dynamic spawning rate our algorithms outperform the original ones, even if co-learning is used with TC-1 and not with TC-HEC and TC-CBG. Also notable is the fact that co-learning seems to have a bigger influence on the TC-1 controller than it has on the other two controllers; actually, co-learning seems to have a negative influence on the TC-CBG controller during rush hour.

5 Conclusion, Discussion and Future Work

From our experiments it can be concluded that the addition of implicit traffic light cooperation through letting cars take into account the traffic situation at the next traffic light has a positive influence on the reinforcement learning algorithm. The algorithms using this new approach seem to always perform better than the original implementation. Also remarkable is the fact that the simpler algorithm, TC-HEC seems to perform at least equally good, but in some cases better, than the more complicated state space expanding TC-CBG. Both are based on the same principle, but TC-CBG is more of a new method, whereas TC-HEC is an added heuristic.

Possible future work is investigating the possibilities of even more global cooperation between traffic light junctions. This can be accomplished by letting a car take into account not only the next traffic light they are visiting, but also the ones that lay further ahead on it's path. An enhancement like this applied to TC-CBG would require the state space to become very large in case of large traffic networks, this might lead to computational problems. TC-HEC on the other hand should be more suitable for dealing with this extra information, since the next traffic lights are incorporated by just multiplying the congestion with the Q values. It might be a good idea to let traffic lights that are further away on it's path have their congestion value discounted by some factor, much like the future V-values are discounted in the reinforcement learning theory.

Whether all this theory can be used in real life situations remains to be seen. As it is now, cars are not yet ready to meet the requirements asked for by this theory; but with the rapid progress in GPS-like technology, it is very well imaginable that in the near future cars will be equipped with technology that is capable of performing the required communication.
Another hurdle in that has to be taken into account is that there are of course many more factors in everyday traffic then just cars and traffic lights. Accidents can happen, getting roads blocked, people can be crossing the street while the light is red, traffic lights can get broken; but at least this theory provides a start towards a new and more sensible traffic light system in big cities.

References

[1] "Intelligent Traffic Light Control", Marco Wiering, Jelle van Veenen, Jilles Vreeken, Arne Koopman, Article, Jul 2004

[2] "Green Light District Simulator", Marco Wiering, http://www.cs.uu.nl/marco, 2003

[3] Tom M. Mitchell, "Machine Learning", McGraw-Hill, chapter 13, pp. 367-390, 1997

[4] R.S. Sutton and A.G. Barto, Reinforcement Learning: An Introduction, The MIT press, Cambridge MA, A Bradford Book, 1998

[5] L.P. Kaelbling, M.L. Littman, and A.W. Moore, " Reinforcement learning: A survey", Journal of Artificial Intelligence Research, vol. 4 pp. 237-285, 1996

[6] G.J. Tesauro, "Temporal difference learning and TD-Gammon", Communications of the ACM, vol. 38, pp. 58-68, 1995

[7] T.L. Thorpe and C. Andersson, "Traffic light control using sarsa with three state representations", Tech. Rep., IBM Cooperation, 1996