
CHALMERS
UNIVERSITY OF TECHNOLOGY

Game Theory ENM140

# Project Report

## Traffic Coordination Game

David Gardtman
920228-4718
cid: gardtman

Simon Nilsson
930128-1854
cid: simnilss

Philip Rasko-Nguyen
920905-4015
cid: pnguyen

My Wester
930614-4206
cid: myve

11th January 2017

# Contents

# 1 Introduction

Throughout the years game theory have proven to be useful in various of different settings besides traditional games. Applying common game theoretic concepts to a certain situation provides tools to build models. These models can then be used for analysis and simulations to find out more about the case that is being studied. In this project, a routing game that has been given the name "The traffic cordination game", which is further described in section 1.1.1, has been studied.

In this report the mentioned game has been analysed theoretically with regards to pure strategies and mixed strategy equilibria (section 2.1). The game has also been modeled in Matlab, using two different approaches. One of them is using a genetic algorithm to study mixed strategies while the other simulation focuses on pure strategies played on a lattice. These simulations are futher described in section 3.

## 1.1 Project definition

This project investigates "The traffic coordination game" defined in section 1.1.1 and discusses its possible uses for traffic situations and other routing applications. To do this a theoretical investigation of the game is performed to understand its mechanics and equilibria. Furthermore, two different simulations, a "genetic simulation" and a "spatial simulation", are used to investigate if and how the different equilibria can occur in the game. In the genetic simulation several rounds of the game is played in a population of players. The mean score is then interpreted as fitness in a genetic algorithm that generates a new population. Finally, in the spatial simulation a number of players are placed on a square grid and each player plays against its neighbourhood. Then each player chooses a new strategy among the players in its neighbourhood.

### 1.1.1 The Traffic Coordination Game

The game is represented by a graph containing exactly two vertices but where the number of edges between them can be varied. Consider a set of $N > 2$ drivers as players. Each driver wishes to go from point A (source vertex) to point B (sink vertex) as fast as possible. Between A and B there are $m$ paths ($1 < m < N$). The paths are ordered in such a way that path 1 is the fastest and path $m$ the slowest. All players must simultaneously choose what path to take, and since driving on a road with lots of cars is slower than driving on an empty one, they are penalised for choosing the same path as other players. The pay-off $P_i$ for choosing path $i$ is given by

$$P_i = m + 1 - i - cN_i, \tag{1}$$

where $N_i$ is the number of other players that chose path $i$ and $c > 0$ is a cost parameter.

## 1.2 Boundaries

This project aims to understand "The traffic coordination game" in order to better understand the mechanics of how traffic queues form. This investigation is restricted to understanding the equilibria that can occur in the game. Depending on the parameters in the game, these theoretical equilbria can be difficult to find. Therefore, the simulations will focus on specific parameters in the game meaning that a complete parameters sweep will not be performed.

# 2 Theoretical investigation

The game that this project revolves around is a routing game we call the "Traffic Coordination Game". It is meant to give a game theoretic description of situations where self interested agents choose between paths of different accesssibility to reach a destination. It can be likened to many coordination games in the sense that the individual pay-off is determined by the actions of the other players since all agents would want to take the fastest road, but if they all do so, there will be less accessibility on the chosen path.

## 2.1 Pure strategy analysis in simple case

Consider now a simple case with $N = 3$, $m = 2$ and $c = 1$. This is one of the simplest, non-trivial cases which allows us to look at the strategic form of the game in a concise and manageable way. That way we can easily respresent the game in strategic form, which can be found in Table 1. Looking at the table, we can see that the best response to the three action profiles is choosing path 2, path 1 and path 1 respectively. All of these choices correspond to the same "player configuration", with two players choosing the first path and one the second path. This is the Nash equilibrium in this simple case.

The result is the same as for the 3-player El Farol Bar game [2], which is not very surprising. Both games share a coordination aspect, as they both could be classified as a type of minority game; where the individual gains from making the same choice as the minority of the players.

Table 1: The table shows the strategic form of the traffic game for $N = 3$, $m = 2$ and $c = 1$. The fourth possible action profile $(2, 1)$ has been omitted here. Since the player roles are symmetric and all players are interchangable it is equivalent to the second action profile in the table, which is the reason it has been omitted.

| | | Action profile of the other players | | |
|---|---|---|---|---|
| | | (1,1) | (1,2) | (2,2) |
| Chosen path | 1 | (0,0,0) | (1,1,1) | (2,0,0) |
| | 2 | (1,1,1) | (0,2,0) | (-1,-1,-1) |

Similarly for the slightly more complex case $N = 4$, $m = 3$ and $c = 1$ we find one Nash equilibrium when 3 players choose the first path and 1 the second. This one is not unique, however, since if one of the players on the first path were to switch to path 3, that would also be a Nash equilibrium. Both of these equilibria are pareto optimal.

Further we claim that in the general case with $N$ players and $m$ paths, every profile in which no path is empty is pareto optimal. This is easily realised as any change in action would result in diminished pay-off for at least one other player.

## 2.2 Pure strategy analysis in the general case

In the general case considering pure strategies there are several interesting properties. Here some of the most interesting properties are presented.

**Property 1:** Players on the same road get the same pay-off.

**Property 2:** If the players are placed in a Nash equilibrium. Then the score difference between the lowest scoring player and the highest scoring player is less than or equal to $c$.

**Property 3:** If there are $N$ players placed in a Nash equilibrium then there is a Nash equilibrium for $N+1$ if the added player is added on the currently highest paying road, or one of the roads tied for the highest paying road.

**Property 4:** If there are $N$ players placed in a Nash equilibrium then there is a Nash equilibrium for $N-1$ if the removed player had the lowest score, or was tied for the lowest score.

**Property 5:** There are at most $\binom{m}{\frac{m}{2}}$ pure strategy Nash equilibria for fixed parameters $N$, $m$ and $c$.

The first property is a result from the pay-off given in equation (1) and proofs for property 2-5 is found in Appendix A.

As result from property 3, to find Nash equilibria for a given number of players $N$, number of roads $m$ and cost parameter $c$ a greedy algorithm cen be defined as follows

1. Place player $j$ at the highest paying road in regards to the previously placed players

2. Repeat until all $N$ players are placed

Any player added according to this rule will be the currently lowest scoring player, a consequence of property 2. Also, since property 4 shows that there is always another Nash equilibrium if the lowest scoring player is removed it is implied that property 3 and 4 are equivalent. If players are repeatedly removed according to the rule in property 4 at some point there will only be one player left. Furthermore, because of property 4 all Nash equilibria can be reduced to the same Nash equilibria, then the greedy rule in the algorithm can be used to find

the Nash equilibria again. This means that the algorithm can find all possible Nash equilibrium by exploring all options when the roads are tied for highest score.

## 2.3 Mixed strategy equilibrium analysis for simple case

If, instead of considering pure strategies, we examine the mixed strategy equilibrium. We will make a slight generalization by keeping the choice of $c$ arbitrary, but for simplicity's sake we will keep $N = 3$ and $m = 2$. As we know, the mixed strategy equilibrium is found when all players are indifferent to their choice in regards to expected pay-off. If all players use the mixed strategy $\sigma = (p, 1-p)$ the expected pay-offs are given by

$$\begin{array}{ll} \langle u(1)\rangle & = p^2(2-2c) + 2p(1-p)(2-c) + 2(1-p)^2 \quad = 2 - 2cp \\ \langle u(2)\rangle & = (1-p)^2(1-2c) + 2p(1-p)(1-c) + p^2 \quad = 1 - 2c + 2cp. \end{array} \tag{2}$$

The equilibrium is then found when $\langle u(1)\rangle = \langle u(2)\rangle$, yielding

$$p = \frac{1+2c}{4c}. \tag{3}$$

For the case earlier with $c = 1$ we see that $p = 3/4$ which is not surprising given a look at the game in strategic form, in Table 1. In three of the four cases (one case omitted in Table 1) it's better to choose the first path, i.e. choosing path 1 with probability 3/4 seems like a good idea.

Looking at Eq. (3) we also find a critical value for $c$ in which $p$ becomes unity. This happens for $c = 1/2$. In this formulation $p$ appears to be larger than unity for $c < 1/2$, but that is simply a limitation of the form it is given in. For $c < 1/2$ the utility for path 1 will always be greater than that of path 2, i.e. path 1 should be chosen with probability 1.

## 2.4 Mixed strategy equilibrium analysis in the general case

In the general case of mixed strategies, because of the symmetry of the game it can be concluded that all players need to play the same mixed strategy to achieve an equilibrium. This strategy can be explained as, choose road $i$ with probability $q_i$. From the analysis in appendix A it can be concluded that given the parameters $N$, $m$ and $c$ the mixed strategy probability $q_i$ that gives an equilibrium is described as

$$q_i = \frac{m(m+1) - 2c(N-1) - 2im}{2mc(N-1)} \tag{4}$$

This formula also gives the critical value of $c$ when there is no mixed strategy equilibrium by setting the smallest probability $q_m$ to zero. This gives the critical value $c > \frac{m(m-1)}{2(N-1)}$.

# 3 Simulations

Part of the examination of the Traffic Coordination game was through simulations. A form of evolutionary simulations were mainly used to study the game. Two different simulation types were used to study mixed strategies and pure strategies respectively. In this section both the simulations themselves and the results obtained from them will be explained.

Matlab was used for both simulations and the code can be found in appendix B as well as at `https://github.com/simnilss/TrafficGame_ENM140`.

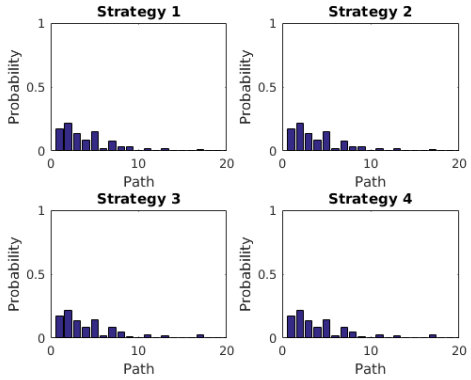
## 3.1 Examining mixed strategies through genetic simulation

The simulation used to study mixed strategies is based on a typical genetic algorithm. The point of this simulation was to have an entire population of randomized mixed strategies and have them evolve over time, to see if the mixed strategy equilibrium would be reached this way, or if some other evolutionary stable strategy would emerge as the most successful. Genetic algorithms will not be explained in detail here. For further reading on the subject we recommend the book Biologically Inspired Optmization Methods by M. Wahde [1]. A very brief description of some of the properties will be given here, however.

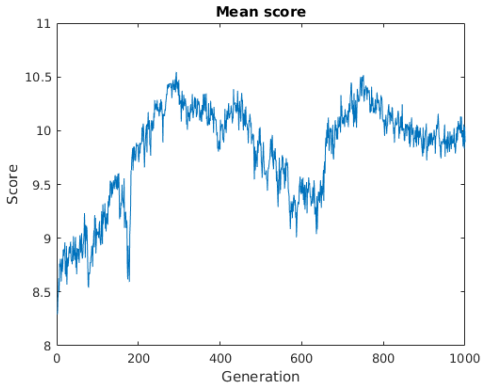
In the simulation, the entire population of $N$ players is participating in the same game. The genetic representation of each strategy is a vector of $m$ real numbers, where $m$ is the number of available paths in the game. Since these are mixed strategies and stochasticity is involved, every generation of players will play $g$ games—typically set to 100. The mean score of every player will then be their *fitness*. With greater fitness, that strategy is more likely to be picked up by a player in the next generation. The best strategy will always survive. Both *crossover* and *mutations* are present in the simulation. *Single-split* crossover is done with probability $p_{cross} = 0.1$ for every pair of selected strategies and *creep mutations* with a creep rate of 0.05 used. The mutation probability is set to be $1/Nm$, so that on average one allele of the genome will be mutated every generation. Any mutations that do occur will with probability $p_{critical} = 0.1$ be a critical mutation, where the new allele value will be independent of the old one. Renormalizations in every generation ensure that the probabilities of the mixed strategies sum to unity.

### 3.1.1 Genetic Simulation Results

Fig. 1 shows results obtained with the simulation explained above. The results are not very surprising. Many runs yield very similar results where the first paths are chosen more likely than the others with a steadily decreasing probability. As long as the number of paths is not too small, there also seems to be some cut-off point where all paths indexed higher than the cut-off are chosen with practically zero probability. Where this cut-off occurs depends on the $m$ and $N$ parameters.




(a) The strategy with the highest score in the last generation (strategy 1) and three random strategies.




(b) The mean score of the population for the different generations.

Figure 1: One result obtained from the genetic simulation described. It is clear that the four strategies shown have conformed quite well towards a similar distribution. It's also worthy of note that paths indexed 10 or higher are chosen with practically zero probability.

This too is not very surprising, since a path should only be considered if lower indexed paths are expected to give the same or lower pay-off.

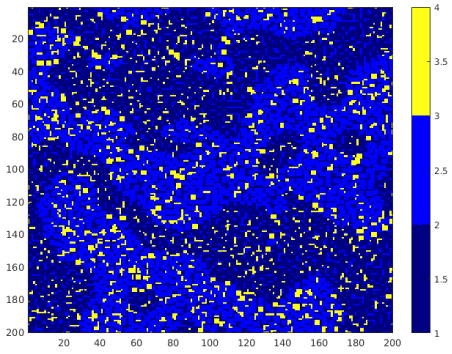
The probability drop-off, for the most part, seems to be somewhat linear. This varies of course, due to the stochasticity of the simulation. A linear probability decrease would be an expected result, since there is a configuration—with linearly decreasing number of players per path—where all players receive the same score. This is the optimal population configuration, in regards to that the mean score will be the highest, as it will match a Nash equilibrium.

Looking at Fig. 1b one notices that the score is not only increasing over time. The decreases are likely the result of oversaturation, where some greedy strategy grew too large, or simply due to randomness. It's also important to note that the score of individuals depend on the whole population. The exact look of this graph changes with different runs. Therefore one must be careful not to draw too many conclusions from a single case.

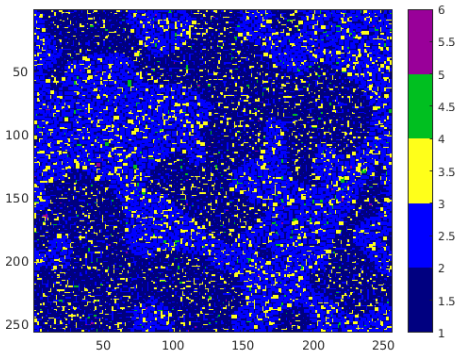
## 3.2 Examining pure strategies through spatial simulation

When examining pure strategies a spatial aspect was introduced. This was done with the hope of being able to find some underlying, previously unidentified, structure in the game. Or if some population-wide coordination/cooperation would emerge from players with very simple strategy updating behaviour.

In this simulation the $N$ players are placed on a square grid of size $L$ ($L^2 = N$) and instead of all players participating in the same game they instead participate in several smaller games with their immediate neighbours. We've chosen to use the von Neumann neighbourhood (including diagonals), so every player has eight neighbours. Consequently, each player will participate in their own game and 8




(a) Spatial result on a 256-by-256 grid with $m = 3$ after 200 timesteps.




(b) Spatial result on 256-by-256 grid with $m = 5$ after 1 000 timesteps.

Figure 2: The two figures show results from the spatial simulation. Both grids are of the same size and the perturbations are the same ($q = 10$), but the number of paths vary. Despite this we see very similar results where path 3 and higher exists in small quantities while path 1 and 2 vastly outnumber them. Additionally, different regions have emerged where there seems to be a clear majority of one of the preferred paths.

others, thus 9 in total.

Over the 9 games each player accumulates a score, which is compared to the accumulated score of the neighbours. Each player will then update their strategy to match the strategy that accumulated the most points in their neighbourhood without preferred direction. This means that if several players obtain the same score, the next strategy will be chosen at random among those players and will be independent of order of updating etc. A player will also always prioritize their own strategy if they are among the highest scoring in the neighbourhood.

To introduce noise, which challenges the stability of strategy configurations, players will with probability $p_s$ switch strategy regardless of their own or their neighbours' score. An alternative way of expressing $p_s$ that makes intuitive sense is

$$p_s = \frac{q}{N}, \tag{5}$$

meaning that the expected number of switches per timestep is exactly $q$.

### 3.2.1 Spatial Simulation Results

Results from simulations on a 256-by-256 grid which has run for 200 and 1 000 timesteps respectively can be found in Fig. 2. The number of path varies, but both simulations have $q = 10$, from Eq. (5). Video results showing the time evolution can be found at `https://github.com/simnilss/TrafficGame_ENM140`. A thing to note in the video results is that only every second timestep is shown, as rapid fluctuations causes a stuttering/flashing effect which is unpleasant to watch and makes it more difficult to se the larger changes over time. This can easily be changed in the code and new video results generated, if the reader wishes.

At first glance it is obvious that path 1 and 2, by far, outnumber path 3 and

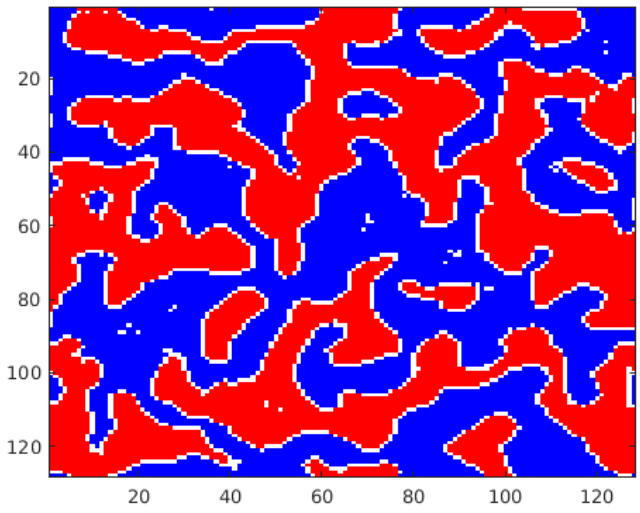



Figure 3: The result from a Schelling model simulation on a 128-by-128 grid (with wrap-around). Agents in this simulation are satisfied if at least 60% of their neighbours are of the same type (red or blue). White tiles are empty.

higher. This is expected behaviour. What is not expected however is the formation of large regions where a lot of players choose the same path. These regions are not homogenous—as can be seen with a closer look—but there is a clear majority that have chosen identically.

This spatial structure is similar in appearance to results from simulating the Schelling model of segregation for sufficiently high satisfaction thresholds, such that segregation occurs. A result from our own implementation can be found in Fig. 3. The code for which (`Schelling.m`) can be found in Appendix B and on the GitHub repository[1].

This would point to some shared underlying structure in the game and the Schelling model or the spatial version of the game and the Schelling model. Looking at the Schelling model from a game theoretic perspective it does, arguably, seem to have a level of coordination built into the behaviour of its agents. This coordination aspect also exists in the traffic game we've devised. One major difference, however, is that the number of players choosing a certain path varies drastically over time in the traffic game, while the number of agents of a certain type in the Schelling model is constant.

The emergence of these regions is likely tied to the chosen update-interaction model, where agents choose the strategy that does best in their neighbourhood. While the game exhibits coordination there are enough differences, we think, for it to not be trivial that this sort of structure would emerge. Looking at the formations in Fig. 2, large parts of them would switch between path 1 and 2 every timestep, which cannot be seen in a single figure nor in the video results as long as only every second timestep is shown. To see it, new results would have to be generated with every timestep shown.

[1] `https://github.com/simnilss/TrafficGame_ENM140`

# 4 Discussion

The simulation results show that the agents exhibit selfish routing behaviour. Selfish routing, choosing the route most optimal for oneself, has been shown to lead to congestion. This can be seen in our simulations as well, where the strategies chosen by players leads to solutions where the total payoff in the system could be higher. In Figure 1a for example, the graph shows a mixed strategy that is not equal to the highest collective pay-off solution. Because of this, one interesting aspect while studying this game is making comparisons to how connected autonomous vehicles could improve traffic throughput.

The solution with the highest total pay-off (the sum of individual payoffs) would be a linear distribution of all players over the road. This is also the solution where the highest possible average payoff is given to all players. A linear distribution therefore translates to the situation where all cars are on the roads between A and B the least amount of time possible. If all drivers could (and would) cooperate, which would be possible with fully connected autonomous vehicles, this solution could be reached. This solution is not reached in our mixed strategy simulation which is to be expected. Because our agents cannot communicate they cannot coordinate. Having only some autonomous vehicles that follow the linear mixed distribution would lead to them being gamed by the other agents that want to maximize their own payoff.

The best average solution might be achievable in the pure strategy simulation. Instead of all players choosing the neighbour with the best average score, there could be some players that choose a road which will average out the scores of the players around them next round. Adding enough of these balancing-players might then lead to all players having the same payoff, which would lead to the players being linearly distributed on all the roads and therefore reaching the best average solution. This means that autonomous vehicles might be able to improve conditions in the game even though they aren't fully connected. In the future, it might be interesting to expand upon this interaction model to get results that are to a greater extent applicable in the real world.

Regarding the supposed connection to the Schelling model, this might also be an interesting topic to study further. So far we've established that the Schelling model shares a coordination aspect with our traffic game but that the spatial aspect—espcially the update-interaction model—is likely a more contributing factor. It is possible that one of the models might be reducible to the other, but so far we've been unable to find the connection—if it exists beyond these results.

Another interesting future application for the game would be to study the effects of the Downs-Thomson paradox. The paradox states that in a network with both car traffic roads and public transport routes, car traffic congestion might actually be increased if another road is added. If more roads are added, improving travel time, more people will want to drive instead of taking public transportation. As a consequence public transit either has to raise their prices or have fewer trips, which also makes people want to drive instead. Expanding this game to

incorporate some representation of this might be an interesting starting point for such a study.

# 5 Conclusions

The "Traffic Coordination Game" has been defined and studied both using simulations and through theoretic analysis. The theoretical analysis has found several properties for the game that are being used as basis for different algorithms that can be applied to the game. Through the results of the simulations, game-theoretic concepts such as selfish routing has been observed. The pure simulation results show some underlying structure of the game that seem to correlate to the Schelling model which indicates that some cooperation occurs between the agents as well. The properties that have been identified simplify the process of finding and identifying nash equilibria. Finally, by using Eq. (4) the mixed strategy equilibrium for any number of roads and players can be found. The simulations of the game seem to exhibit similar results compared to other routing games which would indicate that the game has been represented correctly. As a consequence, we believe that with some modifications it would be possible to use the simulations for studying the Downs-Thomson paradox or the Braess Paradox.

# References

[1] Wahde M. *Biologically Inspired Optimization Methods: An Introduction*. WIT Press, August 2008.

[2] Wikipeda. El farol bar problem – wikipeda, the free encyclopedia. `https://en.wikipedia.org/wiki/El_Farol_Bar_problem`. Accessed 2016-12-01.

# A Proofs

In this appendix contains the proofs for the properties presented in section 2.

## A.1 Property 2

In this game if the players are placed in a Nash equilibrium the score difference between the highest and lowest scoring players is less than $c$, where $c$ is defined in equation (1). To prove this consider that a Nash equilibrium is a situation where no player can achieve a higher score by changing his/her strategy. That means that if a player can better his/her score by switching road then that situation is not an equilibrium. Now consider a situation where the highest scoring player recives $P_{max}$ points, the lowest scoring player recieves $P_{min}$ points and $P_{max} - P_{min} > c$ which implies $P_{min} < P_{max} - c$. If the lowest scoring player switches to the same road as the highest scoring player then according to equation (1) he/she will instead recieve $P_{max} - c$ points which is a better score than $P_{min}$. Thus it can be concluded that $P_{max} - P_{min} < c$ in a Nash equilibrium.

## A.2 Property 3

If there are $N$ players who are placed in a Nash equilibrium, then a new equilibrium can be found for $N + 1$ players by keeping all players on the same road and adding new player to the currently highest paying road. When a new player is placed there are three possibilities if the system is not in an equilibirum.

1. A player can improve his/her score by switching from the selected road

2. A player can improve his/her score by switching to the selected road

3. A player can improve his/her score by switching between two other roads

If the previously placed players were already in an equilibrium then the first scenario wouldn't improve the score for the switching player because the new player choose the highest paying road and since all players on the same road has the same score no player can improve his/her score by switching from that road. Also the second scenario does not improve the switching player's score since no player could improve his/her score by switching to that road before player $N$ was added, and after player $N$ is added that road now has less pay-off. Finally the third scenario does not improve the switching player's score since no player could improve his/her score by switching to that road before player $N$ was added and the payoff for all other roads than the road chosen by player $N$ remains constant. All of these arguments also holds when there are several roads that are tied for the highest pay-off and thus the proof is concluded.

## A.3 Property 4

If there are $N$ players who are placed in a Nash equilibrium, then a new equilibrium can be found for $N-1$ players by removing the currently lowest scoring player. When a player is removed there are three possibilities if the system is not in an equilibirum.

1. A player can improve his/her score by switching from the selected road

2. A player can improve his/her score by switching to the selected road

3. A player can improve his/her score by switching between two other roads

If the remaining players were already place in an equilibrium and the removed player had the lowest score, then the first scenario is not possible beacause of property 1 the selected road is now the road with the highest pay-off. Also the second scenario is not possible since if any player switches to the selected road then again that road will receive the lowest pay-off. Finally, the third scenario does not improve the switching player's score since no player could improve his/her score by switching to that road before player $N$ was added and the payoff for all other roads than the road chosen by player $N$ remains constant. All of these arguments also holds if there are several roads tied for the lowest pay-off. In such a case removing a player from any one of the lowest paying roads gives a Nash equilibrium in the $N-1$ player game. Thus the proof is concluded.

## A.4 Property 5

This property states that there are at most $\binom{m}{\frac{m}{2}}$ pure strategy Nash equilibria for any value of the parameters $N$, $m$ and $c$. To prove this the algorithm from section 2.2 is studied. This algorithm can find all Nash equilibria by exploring all options when there are roads who are tied for highest score. Since there are $m$ roads the configuration that gives most possible Nash equilibria is when there are $m$ roads tied for the highest score. From this configuration, any players added according to the greedy rule has to choose a subset of these $m$ roads without replacement. This means that there are $\binom{m}{k}$ where $k \leq m$ is the number of added players after the configuration with all roads tied for highest score and since $\binom{m}{k} \leq \binom{m}{\frac{m}{2}}$ the proof is concluded.

## A.5 Mixed strategy equilibrium

Because of the symmetry in the game, to reach a mixed strategy equilibrium all players have to play the same strategy. This strategy can be described as "choose road $i$ with probability $q_i$. Using this the expected pay-off from choosing road $i$

can be described as

$$\mathbf{E}[P_i] = \sum_{j=0}^{N-1} \binom{N-1}{j} q_i^j (1-q_i)^{N-1-j}(m+1-i-c\cdot j) \tag{6}$$

Furthermore, to have a mixed strategy equilibrium, the players have to be indifferent to the choice of road which means that $\mathbf{E}[P_i] = D \,\forall i \leq m$ where $D$ is a constant. This condition gives the following

$$\begin{gathered}\sum_{j=0}^{N-1} \binom{N-1}{j} q_i^j (1-q_i)^{N-1-j}(m+1-i-c\cdot j) = D \\ (m+1-i)\sum_{j=0}^{N-1} \binom{N-1}{j} q_i^j (1-q_i)^{N-1-j} - c\sum_{j=0}^{N-1} j\binom{N-1}{j} q_i^j (1-q_i)^{N-1-j} = D\end{gathered} \tag{7}$$

To evaluate this expression consider the sums seperately. The first sum in the expression $\sum_{j=0}^{N-1} \binom{N-1}{j} q_i^j (1-q_i)^{N-1-j}$ is equal to 1. This is can be realised by interpreting it as the cumulative binomial distribution in an experiment with $N-1$ tests and a probability $q_i$ that the test is successful. The sum is then interpreted as the probability that there are less than or equal to $N-1$ successes in the experiment which of course is equal to 1. The second sum $\sum_{j=0}^{N-1} j\binom{N-1}{j} q_i^j (1-q_i)^{N-1-j}$ can be interpreted as the mean value from the same experiment which gives that $\sum_{j=0}^{N-1} j\binom{N-1}{j} q_i^j (1-q_i)^{N-1-j} = (N-1)q_i$. Using this result the previous equation becomes

$$\begin{gathered}(m+1-i)\underbrace{\sum_{j=0}^{N-1} \binom{N-1}{j} q_i^j (1-q_i)^{N-1-j}}_{=1} - c\underbrace{\sum_{j=0}^{N-1} j\binom{N-1}{j} q_i^j (1-q_i)^{N-1-j}}_{=(N-1)q_i} = D \\ (m+1-i) - c(N-1)q_i = D \\ \Rightarrow q_i = \frac{m+1-i-D}{c(N-1)}\end{gathered} \tag{8}$$

Furthermore, since $q_i$ represents the probability of choosing road $i$ it means that

$\sum_{i=0}^{m} q_i = 1$. This gives

$$\begin{aligned}
\sum_{i=0}^{m} q_i &= 1 \\
\sum_{i=0}^{m} \frac{m+1-i-D}{c(N-1)} &= 1 \\
\sum_{i=0}^{m} \frac{m+1-D}{c(N-1)} - \sum_{i=0}^{m} \frac{i}{c(N-1)} &= 1 \\
\frac{m(m+1-D) - \frac{m(m+1)}{2}}{c(N-1)} &= 1 \\
\Rightarrow D &= \frac{m+1}{2} - \frac{c(N-1)}{m}
\end{aligned} \tag{9}$$

which in turn gives

$$\begin{aligned}
q_i &= \frac{m+1-i-D}{c(N-1)} \\
q_i &= \frac{m+1-i-\frac{m+1}{2}-\frac{c(N-1)}{m}}{c(N-1)} \\
q_i &= \frac{m(m+1)-2mi-2c(N-1)}{2c(N-1)}
\end{aligned} \tag{10}$$

# B Simulation code

This appendix contains the Matlab code for the two main simulations mentioned in section 3. It can also be found at `https://github.com/simnilss/TrafficGame_ENM140`.

## B.1 `GeneticSimulation.m`

```
function [finalPopulation] = GeneticSimulation
% ============================================================
% GENETIC SIMULATION FOR GAME THEORY PROJECT
% in the course
% Game Theory and Rationality ENM140, Chalmers
% ============================================================
%
% The simulation features an evoulutionary all-vs-all scenario
% with a population of players playing 'The Traffic Game' for a
% set number of paths and cost parameter.
%
% The 'learning algorithm' chosen is similar to the type of
% evolution found in a standard Genetic Algorithm for
% optimization.
%
% By Simon Nilsson (simnilss)
% Last updated 2016-12-20


% ========== GAME PARAMETERS ============================
populationSize = 50;     % N in mathematical description
nActions = 20;           % m in mathematical description
costParameter = 1;       % c in mathematical description
nRoundsPerGeneration = 100;

% ======== SIMULATION PARAMETERS ==========================
nGenerations = 1000;
crossoverProb = 0.1;
mutationProb = 1/(populationSize*nActions);
criticalMutationProb = 0.1;
mutationRate = 0.05;
% ---------------------------------------------------------

% Set seed based on current time for RNG
rng('shuffle');

% ======= VARIABLES AND 'ALLOCATION' ======================
% INITIAL STRATEGIES
% Generate random initial population strategies
population = NormalizeProbabilities(rand(populationSize, nActions));

% Figures and plot handles
figure(1); clf;
stratPlot = [];
for i=1:min(4, populationSize)
    subplot(2,2,i);
    stratPlot(i) = bar(population(i,:));
    axis([0 nActions 0 1])
    title(['Strategy ' num2str(i)])
    xlabel('Path')
    ylabel('Probability')
end

figure(2); clf;
meanScore = zeros(nGenerations,1);
scorePlot = plot(meanScore);
title('Mean score')
xlabel('Generation')
ylabel('Score')

```

```
% ============= EVOLUTIONARY SIMULATION ==============
% ----------------------------------------------------

bestIndividual = zeros(1, nActions);
for iGeneration = 1:nGenerations
    % ============ PLAY THE GAMES ===============
    score = zeros(1, populationSize);
    for iRound = 1:nRoundsPerGeneration
        % Generate random numbers for path choosing
        r = rand(populationSize, 1);
        % Compute random number thresholds
        thresholds = cumsum(population, 2);


        % Find what paths are chosen
        % ---------------------------
        % This sorts the logical array in descending order and
        % storing the index order for each row, i.e. the index of
        % the first occurrence of a 1 will be in the first column
        % of paths
        % Player i's path choice will be at index i of 'paths'
        [~, pathChoice] = sort(thresholds > repmat(r, [1,size(thresholds,2)]),
            2, 'descend');
        pathChoice = pathChoice(:,1);


        % Count occurrences of path (path(i) in [1,nActions]
        playersOnPath = zeros(nActions, 1);
        for i=1:nActions
            playersOnPath(i) = sum(pathChoice == i);
        end

        % ---- Evaluate scores -----
        for i = 1:populationSize
           score(i) = score(i) + nActions+1-pathChoice(i) ...
               - costParameter*(playersOnPath(pathChoice(i)) - 1);
        end
    end
    % Compute mean score for this generation
    meanScore(iGeneration) = mean(score) / nRoundsPerGeneration;

    % Compute the best result and which player got it
    [~, bestResultIndex] = max(score);
    bestIndividual = population(bestResultIndex,:);

    % Temporarily assign next generation as the current one
    nextGeneration = population;

    % ========== SELECTION AND CROSSOVER ===========
    for i=1:2:populationSize

        % Pick two individuals
        individual_1 = Selection(population, score);
        individual_2 = Selection(population, score);

        % Check if crossover occurs
        if (rand < crossoverProb)
            [individual_1, individual_2] = Crossover(individual_1, individual_2)
                ;
        end

        % Add picked individuals to next generation
        nextGeneration(i,:) = individual_1;

        % Ugly, but without this the population increases by one for odd
        % population sizes
        if (i < populationSize)
            nextGeneration(i+1,:) = individual_2;
        end
    end

    % ========= MUTATION =========
    nextGeneration = Mutate(nextGeneration, mutationProb, ...
```

```
132         mutationRate, criticalMutationProb);
133
134     % Add best individual to population so that it doesn't disappear
135     nextGeneration(1,:) = bestIndividual;
136     % Switch to the new population
137     population = nextGeneration;
138
139
140     % ========== UPDATE PLOTS ===========
141     % Update strategy plot
142     for i = 1:min(4, populationSize)
143         set(stratPlot(i), 'YData', population(i,:));
144     end
145     % Update mean score
146     set(scorePlot, 'YData', meanScore);
147
148     % Arbitrary pause for plots to update
149     pause(0.05)
150
151 end
152
153 % Return final population
154 finalPopulation = population;
155
156 end
157
158 % ============ HELPER FUNCTION DEFINITIONS ===============
159 % -------------------------------------------------------
160
161 function probs = NormalizeProbabilities(P)
162
163     s = sum(P, 2);
164     probs = P ./ repmat(s, [1, size(P,2)]);
165 end
166
167 function [outGene1, outGene2] = Crossover(gene1, gene2)
168
169     minLength = min(length(gene1), length(gene2)) - 1;
170     splitPoint = randi(minLength);
171     outGene1 = [gene1(1:splitPoint) gene2(splitPoint+1:end)];
172     outGene2 = [gene2(1:splitPoint) gene1(splitPoint+1:end)];
173 end
174
175 function outPopulation = Mutate(population, mutProb, rate, criticalMutProb)
176
177     % Generate mutations in the genome
178     mutations = rand(size(population)) < mutProb;
179     % Evaluate how many of them are 'critical mutations'
180     criticalMutations = logical((rand(size(mutations)) < criticalMutProb) .*
            mutations);
181
182     % Carry out mutations and critical mutations
183     outPopulation = population;
184     outPopulation(mutations) = outPopulation(mutations) ...
185         + rate * (2*rand(sum(sum(mutations)), 1) - 1);
186     outPopulation(criticalMutations) = rand(sum(sum(criticalMutations)), 1);
187
188     % Make sure there are no negative probabilities
189     negative = outPopulation < 0;
190     outPopulation(negative) = 0;
191
192     % Renormalize the probabilities
193     outPopulation = NormalizeProbabilities(outPopulation);
194
195 end
196
197 function individual = Selection(population, score)
198
199     % Compute score thresholds for roulette wheel selection
200     scoreThreshold = cumsum(score) / sum(score);
201
202     % Use Roulette-wheel selection
203     r = rand;
```

```
    temp = repmat(r, [1, size(scoreThreshold, 2)]) ...
        < scoreThreshold;
    choice = find(temp, 1);

    individual = population(choice,:);

end
```

## B.2 SpatialSimulation.m

```
function [] = SpatialSimulation
% ====================================================================
% SPATIAL SIMULATION FOR GAME THEORY PROJECT
% Traffic Coordination Game Project, in the course
% Game Theory and Rationality ENM140, Chalmers
% ====================================================================
%
% This simulation add a spatial aspect to the Traffic Coordination
% Game studied. The agents on the grid play only against their
% neighbours, i.e. participate in 9 games (including their own) when
% using the von-Neumann neighbourhood. Optionally the Moore
% neighbourhood can be used.
%
% Only pure strategies are used and the update model is that agents
% choose the strategy (path) of the agent in the neighbourhood with
% the highest score. They will prioritize their own strategy in this.
%
% By Simon Nilsson (simnilss)
% Last update 2016-12-20



% ===== SIMULATION PARAMETERS ================================================
gridSize = 256; % The number of agents will be equal to gridSize*gridSize
nPaths = 3;     % The number of paths in each subgame (1..neighbours-1)
cost = 1;       % The c parameter of the mathematical description
nTimesteps = 1e3;
% The probability that the agent switches path regardless of score
% (to introduce perturbations)
switchProb = 10/(gridSize*gridSize);

% ===== INITIALIZE VARIABLES =================================================
% Initialize a population of gridSize*gridSize agents on a grid with
% random pure strategies represented by integers in the range [1, nPaths]
%population = randi(nPaths, [gridSize*gridSize, 1]);

% OPTION: initialize the entire population with the same strategy
% and let either a single player start with a different one or let
% noise perturbations initiate evolution.
population = ones(gridSize*gridSize, 1);
%population(floor(3*gridSize*gridSize/4)) = 2;

meanScore = zeros(nTimesteps, 1);
customColors =  [0 0 0.5;
                 0 0 1;
                 1 1 0.1;
                 0 0.75 0.125;
                 0.6 0 0.6;
                 0.8 0.1 0.5;
                 1 0 0];

% ----- Create VideoWriter struct and edit settings -----
base = 'spatial sim';
% Change this when running several simulations with the same parameters
nr = 1;
% How many frames should each snapshot take up in the video
nFrames = 2;
filename = strcat(base, ...
                  '-t', num2str(nTimesteps), ...
                  '-L', num2str(gridSize), ...
                  '-m', num2str(nPaths), ...
```

```
                    '-c', num2str(cost), ...
                    '-', num2str(nr), ...
                    '.avi' ...
                    );
video = VideoWriter(filename, 'Uncompressed AVI');
%video.Quality = 100;
open(video);


% ---- Figures ----------------
figure(1); clf;
plotHandle = image( reshape(population,[gridSize,gridSize]) );
% Custom colormap
colormap(customColors(1:nPaths,:));
% Black and white colormap
%colormap('gray')
colorbar

figure(2); clf;
scorePlot = plot(1:nTimesteps, meanScore);
xlabel('Timestep')
ylabel('Mean score')

% ============= SIMULATION LOOP =============
% ------------------------------------------
for t = 1:nTimesteps

    % Let everyone participate in their neighbourhood games
    score = zeros(size(population));
    for i = 1:length(population)
       score = score + SubGame(population, i, gridSize, nPaths, cost);
    end
    meanScore(t) = mean(score);

    % Update everyone's strategy
    newStrategies = zeros(size(population));
    for i = 1:length(population)
        newStrategies(i) = UpdateStrategy(population, score, i, gridSize);
    end
    switches = rand(size(newStrategies)) < switchProb;
    newStrategies(switches) = randi(nPaths, size(newStrategies(switches)));
    population = newStrategies;


    % Update plots
    % ------------
    % NOTE!: But update only every other timestep, for a more 'nice-looking'
    % time evolution. Rapid fluctuations cause visual 'stutter' which is
    % unpleasant to watch.
    if (mod(t, 2) == 0)
        img = reshape(population, [gridSize, gridSize]);
        set(plotHandle, 'CData', img);
        % arbitrary pause for graphics to update
        pause(0.01);
        frame = im2frame(img, customColors);
        for f=1:nFrames
            writeVideo(video, frame);
        end
        set(scorePlot, 'YData', meanScore);
        % arbitrary pause for graphics to update
        pause(0.005);
    end
end

close(video);

end

% ======== HELPER FUNCTION DEFINITIONS =========
% ----------------------------------------------

function score = SubGame(population, coord, gridSize, nPaths, cost)

```

```
    % Find neighbours, with wrap-around
    neighbours = Neighbourhood(coord, gridSize);

    % Extract player strategies and compute path distribution
    strategies = population(neighbours);
    playersOnPath = histcounts(strategies, 0.5:nPaths+0.5)';

    score = zeros(size(population));
    score(neighbours) = nPaths+1 - strategies - cost*(playersOnPath(strategies)
        -1);

end


% Returns a vector containing the indices corresponding to the neighbouring
% tiles in the grid (with wrap-around)
function neighbours = Neighbourhood(coord, gridSize)

    left        = mod( coord-2, gridSize*gridSize ) + 1;
    right       = mod( coord, gridSize*gridSize ) + 1;
    up          = mod( coord-1 - gridSize, gridSize*gridSize ) + 1;
    down        = mod( coord-1 + gridSize, gridSize*gridSize ) + 1;
    upleft      = mod( coord-2 - gridSize, gridSize*gridSize ) + 1;
    upright     = mod( coord - gridSize, gridSize*gridSize ) + 1;
    downleft    = mod( coord-2 + gridSize, gridSize*gridSize ) + 1;
    downright   = mod( coord + gridSize, gridSize*gridSize ) + 1;

    % von-Neumann neighbourhood
    neighbours = [coord left upleft up upright right downright down downleft]';
    % OPTION:
    % Moore neighbourhood
    %neighbours = [coord left up right down]';
end

function strategy = UpdateStrategy(population, score, coord, gridSize)

    neighbours = Neighbourhood(coord, gridSize);

    % Find who got the best scores
    [scores, index] = sort(score(neighbours), 1, 'descend');

    % If player was one of them, keep the current strategy
    if (score(coord) == scores(1))
        strategy = population(coord);
        return;

    % If someone else was better, choose one of those strategies
    else
        % How many got the best score
        nBest = sum(scores == scores(1));
        % Choose one of them at random
        choice = neighbours(index(randi(nBest)));
        strategy = population( choice );
        return;
    end

end
```

## B.3 Schelling.m

```
function [] = Schelling
% ============================================================
% SCHELLING MODEL SIMULATION
% in the course
% Game Theory and Rationality ENM140, Chalmers
% ============================================================
% By Simon Nilsson (simnilss)
% Last updated 2017-01-01

% ================== SIMULATION PARAMETERS ===================
gridSize = 128;
```

```
satisfyThreshold = 0.7;
emptyFrac = 0.1;
nTimesteps = 5e2;

customColors = [1 1 1;
                1 0 0;
                0 0 1];


% --- Initialize agents ---
population = ones(gridSize*gridSize, 1);
population( rand(size(population)) < 0.5 ) = 2;
% Make some tiles empty, so movement is possible
population( rand(size(population)) < emptyFrac) = 0;
emptyIdx = find(population == 0);

% Pre-allocate satisfaction vector
satisfaction = zeros(size(population));


% Set up plot
figure(1); clf;
imgHandle = imagesc( reshape(population, [gridSize,gridSize]) );
colormap(customColors);
colorbar


% ============= SIMULATION LOOP =============
% ------------------------------------------
for iTimestep=1:nTimesteps

    % -- Update the satisfaction of all agents --
    satisfaction(population == 0) = 1;
    for iAgent=1:gridSize*gridSize
        if ( population(iAgent) )
            satisfaction(iAgent) = CheckSatisfaction(iAgent, satisfyThreshold,
                ...
                population, gridSize);
        end
    end

    % -- Move unsatisfied agents --
    [population, emptyIdx] = MoveUnsatisfied(population, satisfaction, emptyIdx)
        ;

    % -- Update graphics in plot --
    img = reshape(population, [gridSize,gridSize]);
    set(imgHandle, 'CData', img);
    % Arbitrary pause for graphics to update
    pause(0.01)

    % Break if no-one is unsatisfied, as nothing will happen
    if (sum(satisfaction == 0) == 0)
        break
    end
end

end

% ================ HELPER FUNCTIONS DEFINITIONS ================
% ---------------------------------------------------------------

% Returns a vector containing the indices corresponding to the neighbouring
% tiles in the grid (with wrap-around)
function neighbours = Neighbourhood(idx, gridSize)

    left        = mod( idx-2, gridSize*gridSize ) + 1;
    right       = mod( idx, gridSize*gridSize ) + 1;
    up          = mod( idx-1 - gridSize, gridSize*gridSize ) + 1;
    down        = mod( idx-1 + gridSize, gridSize*gridSize ) + 1;
    upleft      = mod( idx-2 - gridSize, gridSize*gridSize ) + 1;
    upright     = mod( idx - gridSize, gridSize*gridSize ) + 1;
    downleft    = mod( idx-2 + gridSize, gridSize*gridSize ) + 1;
```

```
    downright   = mod( idx + gridSize, gridSize*gridSize ) + 1;

    % von-Neumann neighbourhood
    neighbours = [left upleft up upright right downright down downleft]';
    % OPTION:
    % Moore neighbourhood
    %neighbours = [left up right down]';
end

function satisfaction = CheckSatisfaction(idx, threshold, population, gridSize)

    agent = population(idx);
    neighbours = population( Neighbourhood(idx, gridSize) );
    % Check how many have the same type
    nSame = sum( neighbours == agent );
    % Obtain fraction of neighbours with same type (ignoring empty tiles)
    fracSame = nSame / sum( neighbours ~= 0 );

    satisfaction = fracSame >= threshold;
end

function [newPopulation, newEmptyIdx] = MoveUnsatisfied(population, satisfaction
    , emptyIdx)

    newPopulation = population;

    % Find unsatisfied agents
    unsatisfied = find(satisfaction == 0);
    nUnsatisfied = length(unsatisfied);
    % Move them one by one
    for i = 1:nUnsatisfied

        % Agent's current location
        agent = unsatisfied(i);

        type = newPopulation(agent);
        newPopulation(agent) = 0;
        % Find new location
        temp = randi(length(emptyIdx));
        newLoc = emptyIdx(temp);
        % Remove index from list of empty ones
        emptyIdx(temp) = [];

        % Move agent to the new location
        newPopulation(newLoc) = type;
        % Add old location to list of empty ones
        emptyIdx = [emptyIdx; agent];

    end

    newEmptyIdx = emptyIdx;

end
```