FH Vorarlberg
University of Applied Sciences

TRAFFYKING

Simulation and Optimisation of Traffic Light Controlled Instersection

Simulation and Optimisation

Vorarlberg University of Applied Sciences
Master's in Mechatronics

Submitted to

Prof. (FH) Dr. Hans-Joachim Vollbrecht

Handed in by

Jan Wenger, Emrah Öztürk, Nicolai Schwartze

Dornbirn, 18.11.2019

Contents

List of Algorithms

1 Task Description

This project is about simulating and optimising a circulative traffic light control. The simulation is done with SUMO which stands for Simulation of Urban MObility. SUMO provides an API that allows external programs to cast and controll a simulation. These control structures, simulation evaluation as well as the optimisation algorithms are programmed in Python 3.7.

The project includes the following tasks:

- create infrastructure to call simulation and alter simulation parametes in SUMO
- construct three different traffic load scenarios
  - Manhatten Grid: 1x1, 1x3, 3x3
  - traffic load: night, noon, rush-hour
- five different optimisation algorithms
  - NSGA2
  - Conjugate Gradinet Descent
  - Differential Evolution
  - self created heuristic using Hill-Climbing
  - self constructed solution
- description and evaluation of the results
- precise technical report to ensure repeatability

The simulation returns three parameters:

- overall waitingtime calculated as the sum of the waiting time of all cars
- overall number of stops calculated as the sum of the stops by all cars
- fairness in waitingtime calculated as the variance of the waitingtimes

We are looking for a solution that minimises all of these parameters.
All of the considered traffic load scenarios are stable. This means, that the waiting line does not grow arbitrarily larg.

2 Simulation Infrastructur

sumo environment, simulation runner, simulation parameter, reroute, tripinfofile and simulation result

Problem: Since the cars can take a turn at every intersection, it might be possible that car gets trapped in the grid for a longer period. This aritficially increases the number of stopcounts and the waitingtime. The optimiser can not improve this situation because the traffic light has no impact on this issue. To prevent that this effect weakens the found solutions, the function evaluation must be adapted. This is done by normalising the waiting time as well as the stopcount with the length of the trip by that specific car. However this also implies, that the grid is equally spaced.

3 Algorithms

This chapter describes the pseudocode and the characteristics of the implemented optimisation algorithms. Alltogether four different optimisers are compared: Differential Evolution (DE), Non Dominated Sorting Genetic Algorithm (NSGA II), Conjugate Gradinet Descent (CGD) and a self-created heuristic based on the Hill-Climbing (HC) principal.

3.1 Differential Evolution

Differential Evolution was first published by Storn and Price 1997. New formes and adaptions are constently developed which belong to some of the best performing optimisation algorithms. The "framework" of Differential Evolution is described in algorithm 3.1.

Algorithm 3.1: Differential Evolution

```
1  population ← initialization
2  while g < G_max do
3      for individual x_i in population do
4          v_i = muation(x_i, population, F)
5          u_i = crossover(x_i, v_i, CR)
6          if function(u_i) < function(x_i) then
7              x_i = u_i
8          end
9          g = g + 1
10     end
11 end
```

The framework allows different definitions of mutation and crossover. In this implementation the mutation $rand\_1$ is used. The following equation 3.1 describes this mutation operator. The subscript i hereby stands for the current individual in the loop. The indizes $r1$ to $r3$ denote random but different individuals in the population.

$$v_i = x_{r1} + F(x_{r2} - x_{r3}) \tag{3.1}$$

The parameter F is also called the scalefactor and can be choosen by the user. Typically this is set to be a number in the intervall $F = [0...1]$.

Further a binomial crossover strategy is used. For every coordinate j in the vector, either an element of the old vector x_i or a new vector v_i is choosen. This again depends on a user defined parameter CR called the crossover rate. To ensure, that at least on single element from the new vector is choosen, a random coordinate K must be transfered.

$$u_{ij} = \begin{cases} v_{ij}, & \text{if } j = K \vee rand[0,1] \leq CR \\ x_{ij}, & \text{otherwise} \end{cases} \tag{3.2}$$

3.2 Nondominated Sorting Genetic Algorithm II

The NSGA-II was proposed by Kalyanmoy Deb et al. 2002. In contraty to the DE, this algorithms is able to optimise multiple functions at the same time. This is called a multicriteria optimiser. The result of this alogorithm are multiple individuals in a population. Ideally this population describes the pareto front. These are all feasible solutions, the user can then decide which optima is best suited.

The implementation was done in conjunction with the algorithm description in appendix B. For a more convenient implementation and readability, every individual in the NSGA II is an object of the class individual which holds the most important attributes of a candidate. The following pseudocode 3.2 outlines the general steps to perform the NSGA II algorithm.

Algorithm 3.2: NSGA II

```
1  population ← initialization
2  while g < G_max do
3      for i in population/2 do
4          p1, p2 = tournament_selection(Pt)
5          q1, q2 = crossover(p1, p2)
6          q1 = mutation(q1)
7          q2 = mutation(q1)
8          Qt = Qt ∩ (q1, q2)
9      end
10     Rt = Rt ∩ (Qt)
11     Rt = fast_non_dominated_sort(Rt)
12     Pt = crowding_distance_sorting(Rt, Pt.size)
13     g = g + 1
14 end
```

3.3 Conjugate Gradient Descent

Compared to the previous algorithms, the Conjugate Gradient Method is a deterministic algorithm which uses a numeric approximation of the gradient. This method is again capable of optimising scalar valued functions.
The original idea dates back to Hestenes and Stiefel 1952. It is specially constructed for optimising quadratic problems of the form $f(x) = \frac{1}{2}x^T Hx + b^T x$. If the problem can be represented in this form, the CGD takes exactly n iteration where n is the dimensionality of the function $f : \mathbb{R}^n \to \mathbb{R}$.
The following pseudocode 3.3 outlines the implemented steps of the CGD. This algorithm also holds two subfunctions: *numGrad* calculates the numeric gradient with the central differencing scheme and the *linsearch* implements a golden section search starting from point x and along a vector d.

Algorithm 3.3: Conjugate Gradient Method

```
1  x, d ← initialization
2  while ||∇f(x)|| < ϵ OR ||η̂d|| < ϵ do
3      grad = numGrad(x)
4      d = −grad + ||grad||² / (||grad_old||² d)
5      if grad d / (||grad|| ||d||) > −α then
6          d = -grad
7      end
8      η̂ = linesearch(f(x, d))
9      x_old = x
10     grad_old = grad
11     x = x + η̂d
12 end
```

3.4 Hill Climbing

The Hill Climbing methode is a very simple from of optimisation algorithm. This method again optimises scalar valued problems. The general idea is to sample points around the current location, and select the new location based on the fitness values of these points. There are two main concepts, that need to be figured out to make the process work are:

- generating new solutions
- step-size adaption

In this implementation, new solutions are produced by going a step in positive and the negative direction of each coordinate. If two consecutive steps do not result in a better solution, the step-size is adapted by halven it. The main issue with this method is, that it performs a strong local search and ultimately leads to a premature convergence.

The following pseudocode 3.4 describes the necessary steps.

Algorithm 3.4: Hill Climbing

```
1  x_start ← initialization
2  while fe < #FE_max do
3      for d in Dim do
4          z[d] = step
5          y_{1,2} = x ± step
6          f = funcion(y)
7          fe = fe + 1
8      end
9      if function(y_d) < function(x) then
10         x = y_d
11     end
12     else
13         step = 0.5 * step
14     end
15 end
```

4 Scenarios

sumo cross, reroute probability (Abbiegewahrscheinlichekit), LP, qin max

5 Experiments

5.1 Results

6 Conclusion

Bibliography

Deb, K. et al. (Apr. 2002): "A fast and elitist multiobjective genetic algorithm: NSGA-II". en. In: *IEEE Transactions on Evolutionary Computation* 6.2, pp. 182–197. ISSN: 1089778X. DOI: 10.1109/4235.996017. URL: http://ieeexplore.ieee.org/document/996017/ (visited on 11/13/2019) (cit. on p. 4).

Hestenes, Magnus R and Stiefel, Eduard (1952): "Methods of Conjugate Gradients for Solving Linear Systems". en. In: p. 28 (cit. on p. 5).

Storn, Rainer and Price, Kenneth (Dec. 1997): "Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces". en. In: *Journal of Global Optimization* 11.4, pp. 341–359. ISSN: 1573-2916. DOI: 10.1023/A:1008202821328. URL: https://doi.org/10.1023/A:1008202821328 (visited on 04/05/2019) (cit. on p. 3).

Appendices

A Task Description (*german*)

Projektbeschreibung

„multikriterielle Optimierung eines umlaufbasierten, ampelgesteuerten Strassenverkehrsnetzes mit Hilfe evolutionärer Algorithmen“

In diesem Projekt soll untersucht werden, wie mit Hilfe evolutionärer Algorithmen eine umlaufbasierte Ampelsteuerung optimiert werden kann. Zu optimierende Kriterien sollen sein:

- Wartezeit
- Anzahl Stopps
- Fairness

Damit handelt es sich also um ein multikriterielles Optimierungsproblem. Zur Optimierung soll dabei (Vorgabe) einmal der bekannte evolutionäre Algorithmus NSGA-II [1], und im Vergleich dazu ein „unikriterieller“ evolutionärer oder genetischer Algorithmus Ihrer Wahl gewählt werden. Letzterer soll die drei obigen Kriterien linear in einer eindimensionalen Zielfunktion optimieren.

Projektziel ist dabei grundsätzlich die Analyse des Optimierungsproblemes zum vertieften Verständnis von umlaufbasierten Ampelregelungen und deren Optimierung durch evolutionäre Algorithmen. Es sollen also synthetische Verkehrsszenarien, nicht reale, definiert und untersucht werden, unter Verfolgung dieses Projektzieles. Zur Bewertung der Leistung der untersuchten beiden Algorithmen soll deshalb auch immer eine manuell (nach „gesundem Menschenverstand“) definierte Vergleichsampelregelung erstellt und bewertet werden.

Folgende Einschränkungen grenzen den Untersuchungsgegenstand ein:

- nur (einheitliche) PKWs, keine Lastwagen, Busse, Anhänger, Motoräder oder Fahrräder
- kein Linksabbiegen bei Gegenverkehr
- nur beampelte Verkehrszusammenflüsse
- nur umlaufbasierte Ampelregelungen
- nur stabile Verkehrsflüsse
- Quellverkehr in das betrachtete System hinein ohne Platoons, reine Exponentialverteilungen der Zwischenankunftszeiten
- kein Fussgängerverkehr
- kein Überholen
- kein adaptives Routing der Verkehrsteilnehmer

Technologische Vorgaben:

- Simulation mit SUMO

Vorgehen:

Grundsätzlich sollten Sie von einfachen zu komplexeren Verkehrsszenarien fortschreiten. Beginnen Sie zum Beispiel mit einer einzigen Kreuzung. Dort können Sie manuell recht einfach eine ordentliche Ampelsteuerung implementieren, da man sich die Auswirkungen verschiedener Umlaufzeiten und Phasen unter verschiedenen Lastszenarien auf die drei obigen Optimierungskriterien noch relativ leicht überlegen kann, unter Berücksichtigung der in der LV vorgestellten Verkehrstheorie. Die evolutionären Algorithmen kann man dann auch entsprechend einfach analysieren.

Dann könnten Sie, mit den gewonnenen Erkenntnissen und einer ersten Implementierung der IT-Infrastruktur (Simulator, Optimierungsalgorithmus, Schnittstelle zwischen beiden, Datensammlung

und Auswertung), zu einer etwas komplexeren Struktur mit mehreren Kreuzungen übergehen (z.B. drei Kreuzungen in Serie). Hier könnte das Hauptziel die Realisierung und die Analyse der Auswirkung einer „grünen Welle“ auf die Optimierungskriterien sein. Zunächst etwa ein Fluss nur in eine Richtung, dann in beide, mit verschiedenen Lastszenarien des die Hauptrichtung querenden Verkehrs etc.

Schliesslich könnten Sie zu einem komplexen Netzwerk (vereinfacht können wir reguläre Gitter wählen) fortschreiten, mit mehreren Verkehrsachsen (mehrere „grüne Wellen“?). Mindestens ein 2x3 (3x3?) Kreuzungsnetz. Auch hier zunächst einmal einen einfachen Fluss (nur geradeaus über die Kreuzungen fahren, Einbahnstrassen) untersuchen, dann Flüsse mit Gegenverkehr, mit Abbiegewahrscheinlichkeiten. Verschiedene Lastszenarien (Haupt- und Nebenverkehrsflüsse) könnten folgen.

Sie sollten bei allen Komplexitätsstufen möglichst drei verschiedene Belastungsstufen wählen: eine leichte (nachts), eine mittlere, und eine hohe. Letztere etwa bei 80%, siehe LV zur Verkehrstheorie. Die Fahrzeugquellen des Verkehrsnetzes sollten mit Zufahrtsraten versehen werden, die nach dem MaxFlow-Algorithmus zu berechnen sind, wie in der LV noch erklärt wird.

Beim Fortschreiten in den Komplexitätsstufen sollte das Verhalten der untersuchten Algorithmen zunehmend klar werden, und damit eine zunehmend rationalere Einstellung der Parameter des jeweiligen Algorithmus.

Bei allem ist auf gute Visualisierung der Ergebnisse zu achten, und vor allem auf Reproduzierbarkeit.

B NSGA II Description

A FAST ELITIST MULTIOBJECTIVE GENETIC ALGORITHM: NSGA-II

ARAVIND SESHADRI

1. Multi-Objective Optimization Using NSGA-II

NSGA ([5]) is a popular non-domination based genetic algorithm for multi-objective optimization. It is a very effective algorithm but has been generally criticized for its computational complexity, lack of elitism and for choosing the optimal parameter value for sharing parameter σ_{share}. A modified version, NSGA-II ([3]) was developed, which has a better sorting algorithm , incorporates elitism and no sharing parameter needs to be chosen *a priori*. NSGA-II is discussed in detail in this.

2. General Description of NSGA-II

The population is initialized as usual. Once the population in initialized the population is sorted based on non-domination into each front. The first front being completely non-dominant set in the current population and the second front being dominated by the individuals in the first front only and the front goes so on. Each individual in the each front are assigned rank (fitness) values or based on front in which they belong to. Individuals in first front are given a fitness value of 1 and individuals in second are assigned fitness value as 2 and so on.

In addition to fitness value a new parameter called ***crowding distance*** is calculated for each individual. The crowding distance is a measure of how close an individual is to its neighbors. Large average crowding distance will result in better diversity in the population.

Parents are selected from the population by using binary tournament selection based on the rank and crowding distance. An individual is selected in the rank is lesser than the other or if crowding distance is greater than the other [1]. The selected population generates offsprings from crossover and mutation operators, which will be discussed in detail in a later section.

The population with the current population and current offsprings is sorted again based on non-domination and only the best N individuals are selected, where N is the population size. The selection is based on rank and the on crowding distance on the last front.

3. Detailed Description of NSGA-II

3.1. **Population Initialization.** The population is initialized based on the problem range and constraints if any.

[1]Crowding distance is compared only if the rank for both individuals are same

3.2. **Non-Dominated sort.** The initialized population is sorted based on non-domination [2]. The fast sort algorithm [3] is described as below for each

- for each individual p in main population P do the following
  - Initialize $S_p = \emptyset$. This set would contain all the individuals that is being dominated by p.
  - Initialize $n_p = 0$. This would be the number of individuals that dominate p.
  - for each individual q in P
    - if p dominated q then
      - add q to the set S_p i.e. $S_p = S_p \cup \{q\}$
    - else if q dominates p then
      - increment the domination counter for p i.e. $n_p = n_p + 1$
  - if $n_p = 0$ i.e. no individuals dominate p then p belongs to the first front; Set rank of individual p to one i.e $p_{rank} = 1$. Update the first front set by adding p to front one i.e $F_1 = F_1 \cup \{p\}$
- This is carried out for all the individuals in main population P.
- Initialize the front counter to one. $i = 1$
- following is carried out while the i^{th} front is nonempty i.e. $F_i \neq \emptyset$
  - $Q = \emptyset$. The set for storing the individuals for $(i+1)^{th}$ front.
  - for each individual p in front F_i
    - for each individual q in S_p (S_p is the set of individuals dominated by p)
      - $n_q = n_q - 1$, decrement the domination count for individual q.
      - if $n_q = 0$ then none of the individuals in the subsequent fronts would dominate q. Hence set $q_{rank} = i + 1$. Update the set Q with individual q i.e. $Q = Q \cup q$.
  - Increment the front counter by one.
  - Now the set Q is the next front and hence $F_i = Q$.

This algorithm is better than the original NSGA ([5]) since it utilize the information about the set that an individual dominate (S_p) and number of individuals that dominate the individual (n_p).

3.3. **Crowding Distance.** Once the non-dominated sort is complete the crowding distance is assigned. Since the individuals are selected based on rank and crowding distance all the individuals in the population are assigned a crowding distance value. Crowding distance is assigned front wise and comparing the crowding distance between two individuals in different front is meaning less. The crowing distance is calculated as below

- For each front F_i, n is the number of individuals.
  - initialize the distance to be zero for all the individuals i.e. $F_i(d_j) = 0$, where j corresponds to the j^{th} individual in front F_i.
  - for each objective function m
    - Sort the individuals in front F_i based on objective m i.e. $I = sort(F_i, m)$.

[2] An individual is said to dominate another if the objective functions of it is no worse than the other and at least in one of its objective functions it is better than the other

- Assign infinite distance to boundary values for each individual in F_i i.e. $I(d_1) = \infty$ and $I(d_n) = \infty$
- for $k = 2$ to $(n-1)$
  - $I(d_k) = I(d_k) + \dfrac{I(k+1).m - I(k-1).m}{f_m^{max} - f_m^{min}}$
  - $I(k).m$ is the value of the m^{th} objective function of the k^{th} individual in I

The basic idea behind the crowing distance is finding the euclidian distance between each individual in a front based on their m objectives in the m dimensional hyper space. The individuals in the boundary are always selected since they have infinite distance assignment.

3.4. **Selection.** Once the individuals are sorted based on non-domination and with crowding distance assigned, the selection is carried out using a ***crowded-comparison-operator*** ($\prec_n$). The comparison is carried out as below based on

(1) non-domination rank p_{rank} i.e. individuals in front F_i will have their rank as $p_{rank} = i$.
(2) crowding distance $F_i(d_j)$

- $p \prec_n q$ if
  - $p_{rank} < q_{rank}$
  - or if p and q belong to the same front F_i then $F_i(d_p) > F_i(d_q)$ i.e. the crowing distance should be more.

The individuals are selected by using a binary tournament selection with crowed-comparison-operator.

3.5. **Genetic Operators.** Real-coded GA's use ***Simulated Binary Crossover (SBX)*** [2], [1] operator for crossover and ***polynomial mutation*** [2], [4].

3.5.1. *Simulated Binary Crossover.* Simulated binary crossover simulates the binary crossover observed in nature and is give as below.

$$c_{1,k} = \frac{1}{2}[(1-\beta_k)p_{1,k} + (1+\beta_k)p_{2,k}]$$
$$c_{2,k} = \frac{1}{2}[(1+\beta_k)p_{1,k} + (1-\beta_k)p_{2,k}]$$

where $c_{i,k}$ is the i^{th} child with k^{th} component, $p_{i,k}$ is the selected parent and β_k (≥ 0) is a sample from a random number generated having the density

$$p(\beta) = \frac{1}{2}(\eta_c + 1)\beta^{\eta_c}, \quad \text{if } 0 \leq \beta \leq 1$$
$$p(\beta) = \frac{1}{2}(\eta_c + 1)\frac{1}{\beta^{\eta_c+2}}, \quad \text{if } \beta > 1.$$

This distribution can be obtained from a uniformly sampled random number u between $(0, 1)$. η_c is the distribution index for crossover[3]. That is

$$\beta(u) = (2u)^{\frac{1}{(\eta+1)}}$$
$$\beta(u) = \frac{1}{[2(1-u)]^{\frac{1}{(\eta+1)}}}$$

[3]This determine how well spread the children will be from their parents.

3.5.2. *Polynomial Mutation.*

$$c_k = p_k + (p_k^u - p_k^l)\delta_k$$

where c_k is the child and p_k is the parent with p_k^u being the upper bound[4] on the parent component, p_k^l is the lower bound and δ_k is small variation which is calculated from a polynomial distribution by using

$$\delta_k = (2r_k)^{\frac{1}{\eta_m + 1}} - 1, \quad \text{if } r_k < 0.5$$

$$\delta_k = 1 - [2(1 - r_k)]^{\frac{1}{\eta_m + 1}} \quad \text{if } r_k \geq 0.5$$

r_k is an uniformly sampled random number between $(0, 1)$ and η_m is mutation distribution index.

3.6. **Recombination and Selection.** The offspring population is combined with the current generation population and selection is performed to set the individuals of the next generation. Since all the previous and current best individuals are added in the population, elitism is ensured. Population is now sorted based on non-domination. The new generation is filled by each front subsequently until the population size exceeds the current population size. If by adding all the individuals in front F_j the population exceeds N then individuals in front F_j are selected based on their crowding distance in the descending order until the population size is N. And hence the process repeats to generate the subsequent generations.

4. Using the function

Pretty much everything is explained while you execute the code but the main arguments to get the function running are population and number of generations. Once these arguments are entered the user would be prompted for number of objective functions and number of decision variables. Also the range for the decision variables will have to be entered. Once preliminary data is obtained, the user is prompted to modify the `objective function`.

Have fun and feel free to modify the code to suit your need!

References

[1] Hans-Georg Beyer and Kalyanmoy Deb. On Self-Adaptive Features in Real-Parameter Evolutionary Algorithm. *IEEE Trabsactions on Evolutionary Computation*, 5(3):250 – 270, June 2001.

[2] Kalyanmoy Deb and R. B. Agarwal. Simulated Binary Crossover for Continuous Search Space. *Complex Systems*, 9:115 – 148, April 1995.

[3] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A Fast Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182 – 197, April 2002.

[4] M. M. Raghuwanshi and O. G. Kakde. Survey on multiobjective evolutionary and real coded genetic algorithms. In *Proceedings of the 8th Asia Pacific Symposium on Intelligent and Evolutionasy Systems*, pages 150 – 161, 2004.

[5] N. Srinivas and Kalyanmoy Deb. Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, 2(3):221 – 248, 1994.

E-mail address: `aravind.seshadri@okstate.edu`

[4]The decision space upper bound and lower bound for that particular component.