AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

FACULTY OF COMPUTER SCIENCE, ELECTRONICS AND TELECOMMUNICATIONS

DEPARTMENT OF COMPUTER SCIENCE

Master of Science Thesis

Implementacja algorytmów dla systemów Monroe i Chamberlina-Couranta z nieliniową funkcją satysfakcji

Implementation of algorithms for Monroe and Chamberlin-Courant systems under nonlinear satisfaction function

Author: *Piotr Szmigielski*
Degree programme: *Computer Science*
Supervisor: *Piotr Faliszewski, PhD*

Kraków, 2016

Oświadczamy, świadomi odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonaliśmy osobiście i samodzielnie (w zakresie wyszczególnionym we wstępie) i że nie korzystaliśmy ze źródeł innych niż wymienione w pracy.

Contents

1. Introduction

We study the effectiveness of algorithms for approximate winner determination under the Monroe [1] and Chamberlin-Courant [2] multiwinner voting rules using nonlinear satisfaction function. Purpose of both the rules is to select a group of candidates that best represent the voters. Having good voting rules and algorithms for them is important, as multiwinner elections are used both in human societies (e.g., for parliament elections) and particular software systems (e.g., in recommendation systems). Rules studied in this paper are exceptionally interesting because they have two desired features of multiwinner rules: they provide accountability (there is a direct connection between the elected candidates and the voters, so each voter has a representative assigned to her and each candidate knows who she represents) and proportional representation of the voters' views.

We assume that candidates participate in the election with multiple winners (a committee with multiple members is selected) and they are elected by voters, each of whom ranks all the candidates (each voter provides a linear order over the set of candidates expressing her preferences). For each voter the Monroe and Chamberlin-Courant rules assign a single candidate as her representative (with some constraints, which are detailed in the further part of the thesis).

The candidates are selected and assigned to the voters optimally, by maximizing the total satisfaction of all the voters. The total satisfaction is calculated as a sum of individual satisfactions of the voters. We assume that there is a satisfaction function that measures how well a voter is represented by the candidate. The function is the same for each voter. It is a decreasing function, so a voter is more satisfied if the candidate assigned to her is ranked higher. In this paper we study cases in which the satisfaction function is a nonlinear one.

The main drawback of the aforementioned rules is that election winner determination is NP-hard under each of them [3] which makes them hard to use in practice, as it would force the use of algorithms that do not provide an optimal result for every data set. Therefore, using these systems for real-life elections may rise some difficulties. However, they can be used for the recommendation systems conveniently, as a good but not optimal recommendation is still useful. Skowron et al. [4] provided approximation algorithms for both the rules that return near-optimal results for various test cases (including real-life data and synthetic data), but for linear satisfaction function only.

In this paper we focus on providing several algorithms for the Monroe and Chamberlin-Courant rules using nonlinear satisfaction function and evaluating them empirically against various data sets. For smaller data, results can be easily assessed by comparing them to the optimal result (calculated with the

brute-force algorithm). For bigger data, the upper bound of the optimal result must be used for comparison. We implement and evaluate various heuristic algorithms, as well as the existing approximation algorithms for the linear satisfaction function, but applying them to the nonlinear cases.

1.1. Motivation and Goals

Monroe and Chamberlin-Courant systems may be potentially very useful, because they are one of the few multiwinner election systems that provide both accountability of candidates to the voters (each voter has one particular representative in the elected committee) and proportional results. Most of the currently used voting rules lack at least one of these properties. For example, D'Hondt method used to elect members of Polish lower house of parliament lacks accountability (voters are accountable to political parties rather than specific parliament members), while single-member constituency plurality system used for United Kingdom parliament elections lacks proportionality.

As finding optimal solution for both of the aforementioned systems is NP-hard [3], there is a need to provide good algorithms which can compute result which is suboptimal but still as close to optimal as possible. Using such result in real-life elections (e.g. for parliament) is disputable. However, for some software applications, e.g. recommendation systems, it can be used seamlessly.

Skowron et. al [4] have already provided approximation algorithms for these systems, but only under linear satisfaction function. In this thesis we focus on providing algorithms for non-linear satisfaction functions, as they can better reflect real preferences of the voters.

1.2. Results

TODO

2. Preliminaries

In the first part of the chapter we explain basic notions regarding multiwinner election problem. Next, we present definitions of winner determination problem under the Monroe and Chamberlin-Courant voting rules. Finally, we present some notions regarding approximate solutions to the problem.

2.1. Basic Notions

Definition 1: Preferences. [4] For each $n \in \mathbb{N}$, by $[n]$ we mean $\{1, \ldots, n\}$. We assume that there is a set $N = [n]$ of *agents* and a set $A = \{a_1, \ldots a_m\}$ of *alternatives*. Each agent i has a strict linear order $\succ_i$ over A, called *preference order* ($a_{\pi(1)} \succ_i a_{\pi(2)} \succ_i \ldots \succ_i a_{\pi(m)}$ for some permutation π of $[m]$). For an alternative a, $pos_i(a)$ represents the position of a in the preference order of agent i. For example, if a is the most preferred alternative for i then $pos_i(a) = 1$, and if a is the least preferred one then $pos_i(a) = m$. A collection $V = (\succ_1, \ldots, \succ_n)$ of agents' preference orders is called a *preference profile*.

If A is the set of all the alternatives and B is a nonempty strict subset of A, then by $B \succ A - B$ we mean that all alternatives in B are preferred to those outside of B for the preference order $\succ$.

Definition 2: Positional scoring function. [4] A *positional scoring function* (PSF) is a function $\alpha^m : [m] \to \mathbb{N}$. It assigns a score to every position in the agent's preference order. If for each $i, j \in [m]$, if $i < j$ then $\alpha^m(i) > \alpha^m(j)$, a PSF α^m is a *decreasing positional scoring function* (DPFS) .

If α^m is a DPSF then $\alpha^m(i)$ can be used to represent the *satisfaction* value of an agent when she is assigned to an alternative that is i'th in her preference order. For every DPSF $\alpha^m(m) = 0$, so an agent is not satisfied at all with her worst alternative. In some cases, we will write α instead of α^m to simplify notation.

A DPSF α^m is considered a *linear satisfaction function* if for each $i \in [m-1]$ value $\alpha^m(i+1) - \alpha^m(i)$ is constant. Every DPSF not satisfying that constraint is considered a *nonlinear satisfaction function*.

We define a family α of DPSFs as $\alpha = (\alpha^m)_{m\in\mathbb{N}}$, where α^m is a DPSF on $[m]$, such that $\alpha^{m+1}(i+1) = \alpha^m(i)$ for all $m \in \mathbb{N}$ and $i \in [m]$. Families of DPSFs are built iteratively by prepending values to functions with smaller domains (smaller m), leaving existing values from the previous function (previous family member) unchanged. Such families of DPSFs are called *normal* DPSFs.

Definition 3: Assignment functions. [4] We define a *K-assignment function* as any function $\Phi : N \to A$ that satisfies $\|\Phi(N)\| \leq K$ (it assigns agents to no more than K alternatives). A *Monroe K-assignment function* is a K-assignment function that additionally satisfies the following constraint: For each alternative $a \in A$ we have that either $\left\lfloor \frac{\|N\|}{K} \right\rfloor \leq \|\Phi^{-1}(a)\| \leq \left\lceil \frac{\|N\|}{K} \right\rceil$ or $\|\Phi^{-1}(a)\| = 0$. It means that for Monroe K-assignment function, agents are assigned to exactly K alternatives and each of the alternatives has about $\frac{\|N\|}{K}$ agents assigned.

A *partial K-assignment function* is similar to a K-assignment function, with one exception: it may assign an empty alternative, $\perp$, to the agents. A *partial Monroe K-assignment* is a partial K-assignment that can be extended to a regular Monroe K-assignment by replacing each assignment of an empty alternative with one from A. If we have an assignment function Φ, alternative $\Phi(i)$ is called the *representative* of agent i.

Let S be a set of alternatives. By Φ^S we mean a (possibly partial) K-assignment function that assigns agents only to alternatives from S.

Definition 4: Total satisfaction function. [4] We assume that α is a normal DPSF. Following function assigns a positive integer to a given assignment Φ:

$$l^{\alpha}_{sum}(\Phi) = \sum_{i=1}^{n} \alpha(pos_i(\Phi(i))) \tag{2.1}$$

This function combines satisfaction of the agents to assess the quality of the assignment for the entire society. It simply calculates the sum of the individual agents satisfaction value and is used as the *total satisfaction function*.

For each subset of the alternatives $S \subseteq A$ that satisfies $\|S\| \leq K$, Φ^S_α denotes the partial (Monroe) K-assignment that assigns agents only to the alternatives from S and such that Φ^S_α maximizes the total satisfaction $l^{\alpha}_{sum}(\Phi^S_\alpha)$.

2.2. Monroe and Chamberlin-Courant Rules

We will now define the problems of winner determination under the Monroe and Chamberlin-Courant (abbreviated as CC) rules. The goal is to find an optimal assignment function, where by the

optimal function we accept one that maximizes the total satisfaction.

Definition 5: SatWinner. [4] Let α be a normal DPSF. An instance of α-*CC-SatWinner* problem is defined as follows: given a set of agents $N = [n]$, a set of alternatives $A = \{a_1, \ldots, a_m\}$, a preference profile V of the agents, and positive integer K representing committee size, we seek a K-assignment function Φ that maximizes $l^{\alpha}_{sum}(\Phi)$. The problem α-*Monroe-SatWinner* is defined in the same way but we additionally require Φ to be a Monroe K-assignment function.

These problems' intention is to find a (Monroe) K-assignment function which returns a set of K alternatives, who are viewed as the winners of the given multiwinner election (e.g. elected members of a committee).

2.3. Approximate Solutions

As for many normal IPSFs multiwinner election problems under both Monroe and Chamberlin-Courant rules are NP-hard [3, 5], we are looking for approximate solutions.

Definition 6: Approximation algorithms. [4] Let r be a real number such that $0 < r \leq 1$, let α be a normal DPSF, and let R be either Monroe or CC. An algorithm is an *r-approximation algorithm* for α-R-SatWinner problem if on each instance I it returns a feasible assignment Φ such that $l^{\alpha}_{sum}(\Phi) \geq r \cdot OPT$, where OPT is the optimal total satisfaction $l^{\alpha}_{sum}(\Phi_{OPT})$.

3. State of the Art

4. Implemented Algorithms

In this chapter we present implemented algorithms for the utilitarian versions of Monroe and Chamberlin-Courant multiwinner voting rules in the satisfaction-based framework.

Proposition 1 (Implicit in the paper of Betzler et al. [5]). Let α be a normal DPSF, N be a set of agents, A be a set of alternatives, V be a preference profile of N over A, and S a K-element subset of A (where K divides $\|N\|$). Then there is a polynomial-time-algorithm that computes a (possibly partial) optimal K-assignment Φ_α^S (Monroe K-assignment Φ_α^S) of the agents to the alternatives from S.

4.1. Existing Algorithms

In this section we present algorithms that already exist, but have never been applied to nonlinear cases before.

4.1.1. Algorithms A, B and C

Algorithm A was first presented by Skowron et al. [4] and tries to solve α-Monroe-SatWinner. It builds a solution iteratively (greedily). In every iteration we pick some alternative a_i that have not been assigned yet and assign it to those $\frac{N}{K}$ agents that are not assigned to any other alternative yet and whose satisfaction of being matched with a_i is maximal (criterion for picking an alternative in each step is a sum of satisfaction of agents selected this way). This algorithm runs in polynomial time [4]. Pseudocode is presented in Algorithm 1.

Algorithm B is an extension to Algorithm A and was presented in the same paper. The idea is to run Algorithm A first and then optimally reassign the alternatives to the voters (using Proposition 1). It should noticeably improve the results of the algorithm and it still runs in polynomial time.

Algorithm C is a further extension to Algorithm B, also presented by Skowron et al. [4]. While Algorithm B only keeps one partial assignment function Φ that is extended in each step until it becomes a full assignment, Algorithm C stores a list of d functions (d is provided as an algorithm parameter).

Algorithm 1 Algorithm A

procedure COMPUTEMONROESATWINNER
  $\Phi \leftarrow$ a map defining a partial assignment, iteratively built by the algorithm
  $\Phi^{\leftarrow} \leftarrow$ the set of agents for which the assignment is already defined
  $\Phi^{\rightarrow} \leftarrow$ the set of alternatives already used in the assignment
  if $K \leq 2$ **then**
    compute the optimal solution using an algorithm of Betzler et al. [4] and return
  Φ = {}
  for $i \leftarrow 1$ to K **do**
    $score \leftarrow \{\}$
    $bests \leftarrow \{\}$
    for all $a_i \in A \setminus \Phi^{\rightarrow}$ **do**
      $agents \leftarrow$ sort $N \setminus \Phi^{\leftarrow}$ so that if agent j preceeds agent j' then $pos_j(a_i) \leq pos_{j'}(a_i)$
      $bests[a_i] \leftarrow$ choose first $\frac{N}{K}$ elements from $agents$
      $score[a_i] \leftarrow \sum_{j \in bests[a_i]}(m - pos_j(a_i))$
    $a_{best} \leftarrow argmax_{a \in A \setminus \Phi^{\rightarrow}} score[a]$
    for all $j \in bests[a_{best}]$ **do**
      $\Phi[j] \leftarrow a_{best}$

At each step, for each alternative a with no agent assigned and for each Φ of the d functions stored, we compute an extension to Φ that assigns $\frac{N}{K}$ agents (that are not assigned to any other alternative yet) to a (the same way as in Algorithm A). For the next step, d functions that give the highest satisfaction are kept. If we take $d = 1$, we obtain Algorithm B. Pseudocode is presented in Algorithm 3.

Unlike previous algorithms, Algorithm C can be used for both Monroe and Chamberlin-Courant rules. To adapt it to the Chamberlin-Courant rule, we have to replace the contents of the first for all loop with the appropriate code, presented in Algorithm 2.

Algorithm 2 Algorithm C - CC for all code replacement

for all $a_i \in A \setminus \Phi^{\rightarrow}$ **do**
  $\Phi' \leftarrow \Phi$
  for all $j \in N$ **do**
    if agent j prefers a_i to $\Phi'(j)$ **then**
      $\Phi'(j) \leftarrow a_i$
  $newPar.push(\Phi')$

Algorithm 3 Algorithm C

procedure COMPUTEMONROESATWINNER
    $\Phi \leftarrow$ a map defining a partial assignment, iteratively built by the algorithm
    $\Phi^{\leftarrow} \leftarrow$ the set of agents for which the assignment is already defined
    $\Phi^{\rightarrow} \leftarrow$ the set of alternatives already used in the assignment
    $Par \leftarrow$ a list of partial representation functions
    $Par = []$
    $Par.push(//)$
    for $i \leftarrow 1$ to K **do**
        $newPar = []$
        for $\Phi \in Par$ **do**
            $bests \leftarrow \{\}$
            for all $a_i \in A \setminus \Phi^{\rightarrow}$ **do**
                $agents \leftarrow$ sort $N \setminus \Phi^{\leftarrow}$ (agent j preceeds agent j' implies that $pos_j(a_i) \leq pos_{j'}(a_i)$
                $bests[a_i] \leftarrow$ choose first $\frac{N}{K}$ elements of $agents$
                $\Phi' \leftarrow \Phi$
                for all $j \in bests[a_i]$ **do**
                    $\Phi'[j] \leftarrow a_i$
                $newPar.push(\Phi')$
        sort $newPar$ according to descending order of the total satisfaction of the assigned agents
        $Par \leftarrow$ choose first d elements of $newPar$
    for $\Phi \in Par$ **do**
        $\Phi \leftarrow$ compute the optimal representative function using an algorithm of Betzler et al. [5] for the set of winners $\Phi^{\rightarrow}$
    return the best representative function from Par

4.1.2. Algorithm R

As shown by Skowron et al. [4], algorithms A, B and C achieve very high approximations ratios under linear satisfaction function for the cases where K is small relative to m. For the other cases, we can use a sampling-based randomized algorithm (called Algorithm R). We expect that under nonlinear satisfaction function algorithms should behave similarly in relation to each other as under a linear one.

Algorithm R randomly picks K alternatives and then, using Proposition 1, assigns them to agents optimally. As such an algorithm may be simply unlucky and pick alternatives that are ranked low, random assignment should be computed a given number of times (which is provided as a parameter), so there is a greater probability to attain a high quality solution. If K is comparable to m then it is likely that generated results would include a solution that is at least close to optimal. Algorithm can naturally be used for both Monroe and Chamberlin-Courant systems.

4.1.3. Algorithm AR

Algorithm family A-C and Algorithm R are naturally suitable for different cases. Therefore, Skowron et al. [4] proposed to combine algorithms A and R into Algorithm AR. They also showed that under linear satisfaction function, Algorithm AR can achieve approximation ratio of $0.715 - e$ with probability λ. Both e and λ are provided as algorithm parameters. Naturally, for different satisfaction functions approximation ratio may vary, but we decided to test the algorithm in an unchanged version for comparison. Pseudocode is presented in Algorithm 4.

Algorithm 4 Algorithm AR

procedure COMPUTEMONROESATWINNER
  H_j is the j'th harmonic number $H_j = \sum_{i=1}^{j}(\frac{1}{i})$
  $\lambda \leftarrow$ probability of achieving the approximation ratio $0.715 - e$ under linear satisfaction function
  if $\frac{H_K}{K} \geq \frac{e}{2}$ **then**
    compute the optimal solution using an algorithm of Betzler et al. [5] and return
  if $m \leq 1 + \frac{2}{e}$ **then**
    compute the optimal solution using a simple brute force algorithm and return
  $\Phi_1 \leftarrow$ solution computed by Algorithm A
  $\Phi_2 \leftarrow$ solution computed by Algorithm R (sampled $\log(1-\lambda) \cdot \frac{2+e}{e}$ times)
  return the better assignment among Φ_1 and Φ_2

4.1.4. Algorithm GM

Algorithm GM (greedy marginal improvement) is an algorithm that was introduced by Lu and Boutilier [6] for the Chamberlin-Courant rule only. However, it was later generalized by Skowron et al. [4], so it can be applied to the Monroe rule as well. In the Monroe case, it can be considered as the

Algorithm B improvement.

We start with an empty set S. In each iteration of the algorithm we select an alternative a that is not assigned to agents yet, and that maximizes the satisfaction value $l^{\alpha}_{sum}(\Phi^{S\cup\{a\}}_{\alpha})$. Iterations are executed until a complete committee is selected, so K iterations are required. For Monroe case, computing Φ^{S}_{α} is slow (it is achieved by using min-cost/max-flow algorithm [5]), which makes the algorithm execute for a relatively long time. Pseudocode is presented in Algorithm 5.

Algorithm 5 Algorithm GM

1: **procedure** COMPUTESATWINNER
2: Φ^{S}_{α} - the partial assignment that assigns a single alternative to at most $\frac{n}{K}$ agents, that assigns to the agents only the alternatives from S, and that maximizes the utilitarian satisfaction $l^{\alpha}_{sum}(\Phi^{S}_{\alpha})$
3: $S \leftarrow \emptyset$
4: **for** $i \leftarrow 1$ to K **do**
5: $a \leftarrow argmax_{a\in A\setminus S} l^{\alpha}_{sum}(\Phi^{S\cup\{\alpha\}}_{\alpha})$
6: $S \leftarrow S \cup \{a\}$
7: **return** Φ^{S}_{α}

4.1.5. Algorithm P

Algorithm P can only be applied to Chamberlin-Courant problem. It was first introduced by Skowron et al. [4]. In the beginning, it computes x (a non-negative integer). Next, it computes an assignment that should maximize the number of agents that have an alternative from the first x spots in their preferences assigned to them. This process is executed greedily. Afterwards, if there are still agents with no alternative assigned, the best alternative is picked from the ones already selected for at least one other agent.

$w(x)$ used in the algorithm is a Lambert's W-function, defined to be the solution of the equation $x = w(x)e^{w(x)}$. Algorithm runs in polynomial time. Pseudocode of Algorithm P is presented in Algorithm 6.

4.2. New Algorithms

In this section we present algorithms that have been invented and implemented specifically for the purpose of this thesis.

Algorithm 6 Algorithm P

procedure COMPUTECCSATWINNER
  $\Phi \leftarrow$ a map defining a partial assignment, iteratively built by the algorithm
  $\Phi^{\leftarrow} \leftarrow$ the set of agents for which the assignment is already defined
  $\Phi^{\rightarrow} \leftarrow$ the set of alternatives already used in the assignment
  $num\_pos_x(a) \leftarrow \|\{i \in [n] \setminus \Phi^{\leftarrow} : pos_i(a) \leq x\}\|$ - the number of not-yet assigned agents that rank alternative a in one of their first x positions
  $w(\cdot)$ - Lambert's W-function
  $\Phi = \{\}$
  $x = \left\lceil \frac{mw(K)}{K} \right\rceil$
  for $i \leftarrow 1$ to K **do**
    $a_i \leftarrow argmax_{a \in A \setminus \Phi^{\rightarrow}} num\_pos_x(a)$
    for all $j \in [n] \setminus \Phi^{\leftarrow}$ **do**
      if $pos_j(a_i) < x$ **then**
        $\Phi[j] \leftarrow a_i$
  for all $j \in A \setminus \Phi^{\leftarrow}$ **do**
    $a \leftarrow$ such alternative from $\Phi^{\rightarrow}$ that $\forall_{a' \in \Phi^{\rightarrow}} pos_j(a) \leq pos_j(a')$
    $\Phi[j] \leftarrow a$
  return Φ

4.2.1. Genetic Algorithm

The idea of our algorithm is loosely based on metaheuristic genetic algorithms, such as the Firefly Algorithm [7].

Our Genetic Algorithm starts with an initial set of creatures (each of them presenting a possible solution under Chamberlin-Courant or Monroe rule) which are later mutated and crossed over with each other. The best creatures (ones with the highest total satisfaction) are preferred for further mutation and crossover in order to better investigate the neighbourhood of local maxima, but on the other hand algorithm also produces new creatures by crossing over random existing ones to better explore entire solution space, not limiting to local extrema.

In each iteration of the algorithm creatures are evaluated (satisfaction is computed). Best creature in terms of total satisfaction is compared with currently best found creature and takes its place if it is better. Half of the evaluated creatures (the best ones) are chosen for further propagation. Each of them is then mutated randomly. Remaining creatures are created by crossing over random creatures from the "better" half with each other. Resulting set of creatures is used for the next iteration. Number of iterations and number of creatures are the algorithm parameters. Pseudocode is presented in Algorithm 7.

Algorithm 7 Genetic Algorithm

procedure COMPUTECCSATWINNER
  I - number of iterations
  c - number of creatures
  Φ_{best} - best creature (preference profile)
  $creatures \leftarrow$ generate initial random set of c creatures
  for $i \leftarrow 1$ to I **do**
    $creaturesSorted \leftarrow$ sort $creatures$ by total satisfaction
    if $satisfaction(creaturesSorted[1]) > satisfaction(\Phi_{best})$ **then**
      $\Phi_{best} = creaturesSorted[1]$
    $bestCreatures \leftarrow$ choose first $c/2$ elements from $creaturesSorted$
    $mutated \leftarrow$ mutate all creatures from $bestCreatures$ randomly
    $crossed \leftarrow$ crossover random creatures from $bestCreatures$ to produce $c/2$-element set
    $newCreatures \leftarrow mutated \cup crossed$
    $creatures \leftarrow newCreatures$
  return Φ_{best}

4.2.2. Simulated Annealing

The Simulated Annealing algorithm is inspired by physical annealing process, used in metallurgy. It involves heating and cooling a material to make it attain specific physical properties. Using simulated annealing for optimization of a complex function depending on many parameters was proposed by Kirkpatrick et al. [8]. We adapt it to the Chamberlin-Courant and Monroe problems.

In simulated annealing we use a temperature variable, which has a high value at the beginning and gets lower ('cools') during the execution. When the temperature is high, algorithm can accept solutions worse than the current one more frequently, so it is possible to leave a local optimum, as the global one may be in totally different area of the search space. When the temperature gets lower, algorithm focuses on the area where solution close to the optimum may lie.

To decide if the new solution should be accepted, the *acceptance function* is used. If the new solution is better, it is always accepted. If it is worse, acceptance function accepts the new solution with probability p, which is calculated as follows:

$$p = \exp(\frac{E_c - E_n}{T}) \tag{4.1}$$

E_c is the current solution energy, E_n is the new solution energy and T is the temperature. Energy represents quality of the solution and, in our case, is proportional to the total satisfaction value of the solution.

The algorithm proceeds as follows. First, it generates a random initial solution. Initial temperature is given as a parameter. Then, in each iteration, a new solution is generated by replacing one of the winners in the current solution with a random alternative that is not a winner in the current solution. The newly created solution is then evaluated by the acceptance function. If it is accepted, it replaces the current solution. Otherwise, the current solution is kept. For the next iteration, temperature is decreased:

$$T \leftarrow T \cdot (1 - c) \tag{4.2}$$

c is the cooling rate, provided as a parameter. The algorithm stops when $T \leq 1$. Pseudocode is presented in Algorithm 8.

Algorithm 8 Simulated Annealing

procedure COMPUTECCSATWINNER
    T_{start} - initial temperature
    c - cooling rate
    Φ_{curr} - current solution
    T - current temperature
    $E(\Phi)$ - energy of solution Φ
    $\Phi_{curr} \leftarrow$ generate initial random solution
    $T \leftarrow T_{start}$
    while $T > 1$ **do**
        $\Phi_{new} \leftarrow$ perform random replacement on Φ_{curr}
        $p \leftarrow \exp(\frac{E(\Phi_{curr}) - E(\Phi_{new})}{T})$
        $r \leftarrow$ generate random number in range $[0; 1)$
        if $E(\Phi_{new}) > E(\Phi_{curr})$ or $p > r$ **then**
            $\Phi_{curr} \leftarrow \Phi_{new}$
        $T \leftarrow T \cdot (1 - c)$
    return Φ_{curr}

5. Evaluation Results

6. Summary

References

[1] B. Monroe. Fully proportional representation. *American Political Science Review*, 89(4):925–940, 1995.

[2] B. Chamberlin and P. Courant. Representative deliberations and representative decisions: Proportional representation and the borda rule. *American Political Science Review*, 77(3):718–733, 1983.

[3] A.D. Procaccia, J.S. Rosenschein, and A. Zohar. On the complexity of achieving proportional representation. *Social Choice and Welfare*, 30(3):353–362, apr 2008.

[4] P. Skowron, P. Faliszewski, and A. Slinko. Achieving fully proportional representation, approximability results. *Artificial Intelligence*, 222:67–103, may 2015.

[5] N. Betzler, A. Slinko, and J. Uhlmann. On the computation of fully proportional representation. *Journal on Artificial Intelligence Research*, 47:475–519, 2013.

[6] T. Lu and C. Boutilier. Budgeted social choice: From consensus to personalized decision making. *IJCAI*, 11:280–286, 2011.

[7] S. Łukasik and S. Żak. Firefly algorithm for continuous constrained optimization. *Lecture Notes in Artificial Intelligence*, 5796:97–106, 2009.

[8] S. Kirkpatrick, C.D. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

List of Tables