Chapter 19
Does This Suit Me? Validation of Self-modeling Network Models by Parameter Tuning

Jan Treur

Abstract In this chapter it is discussed how a personalised temporal-causal network model can be obtained that fits well to specific characteristics of a person, and his or her connections and further context. A model is an approximation, but always a form of abstraction of a real-world phenomenon. Its accuracy and correctness mainly depend on the chosen abstracting assumptions and the personal and contextual (network) characteristics defining the model. Depending on the complexity of the model, the number of its characteristics can vary from just a couple to thousands. These network characteristics usually represent specific features or properties of the modelled phenomenon, for example, for modelling human processes personality traits or social interaction properties. No values for such characteristics are given at forehand. From a more general and abstract view, they can be considered parameters of the model. Estimation of such parameters for a given model is a nontrivial task. In this chapter, it is discussed how this can be addressed for temporal-causal network models based on the parameter tuning method of Simulated Annealing and a specific component within the dedicated modeling environment, thereby making use of MATLAB's built-in optimser Optimtool.

Keywords Validation · Self-modeling network models · Parameter tuning · Simulated annealing · Root mean square error

19.1 Introduction

For computational models in general, an important issue is how it can be justified that the model is valid for the real-world situation or phenomenon that is modeled. A process to answer this question is called *validation*. This also applies to network models. One form of validation can be obtained from the way in which a network model is designed:

J. Treur (✉)
Social AI Group, Department of Computer Science, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands
e-mail: j.treur@vu.nl

J. Treur and L. Van Ments (eds.), *Mental Models and Their Dynamics, Adaptation, and Control*, Studies in Systems, Decision and Control 394,
https://doi.org/10.1007/978-3-030-85821-6_19

- To model some phenomenon you study, you determine which underlying mechanisms make the phenomenon happen
- For this you use empirically founded knowledge or theories from relevant scientific disciplines
- This provides a scientific justification of the structure of the network model covering these mechanisms
- This can be reported in any form of documentation or presentation of the model.

In addition to this, after the model has been designed further validation can be performed:

- By simulation of the model it has to be shown that indeed these mechanisms bring about the studied real-world phenomenon
- For this, settings for the model's network characteristics are needed that fit well to the context that is modeled, for example, a specific type of person
- Choosing adequate values for these network characteristics is not always easy
- To support the search for adequate values, automated parameter tuning methods can be helpful.

The use of such an automated parameter tuning method is the main topic of the current chapter. The general picture of this is obtained as a typical classical AI approach involving a search space and a search method; this goes as follows.

Dynamical models such as temporal-causal network models usually have to take into account a number of *situational characteristics* of the real-world situation that is modelled. Such situational characteristics can involve, for example, the mental or neurological structures of a person, or a person's connections to others, or contextual elements of the external world. Within a network model certain *network characteristics* (such as connection weights, speed factors, combination function weights, and combination function parameters) are used to represent such situational characteristics. The advantage of having such network characteristics in a network model description is that they enable to use and tune the network model for different situations: for example, persons with different mental or neurological structures, for different social connections, or for different contextual elements in the external world.

In fact, the model represents a large (and in theory infinite) space of possibilities indicated by all combinations of values of the network characteristics. For example, suppose 10 network characteristics are involved and all of them have values in the interval [0, 1]. If only values in one digit are considered (i.e., 0, 0.1, …, 0.9, 1.0), then the number of combinations already is 11^{10}, which is more than 25 billion or $2.5 * 10^{10}$. If the values are considered in two digits, this number will be more than 100^{10}, which is 10^{20}. So, a model with a large number of network characteristics is very generic in the sense that the space of situations that can be represented by the model can be huge, with many variations.

For one given specific situation at hand the network characteristics have to be assigned values that represent that situation in particular; values have to be found that fit to the situation: by finding such values, knowledge of the specific situational

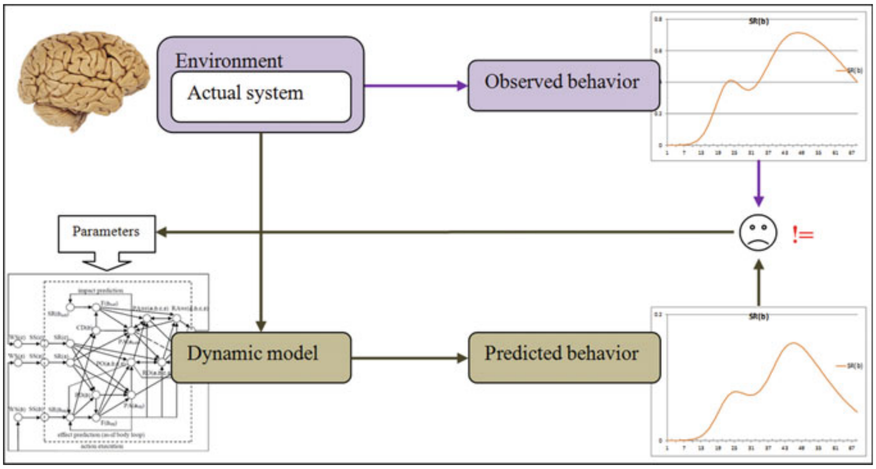


Fig. 19.1 Parameter estimation for a dynamical model

characteristics of the situation is acquired. However, acquisition of specific characteristics is not always easy, as often a situation that is modelled does not simply show these characteristics. They have to be acquired or estimated by some process in one way of the other, and this may turn out to be not so easy and often has to be done in an indirect manner via a specific search method applied to the search space of all possibilities. In Fig. 19.1 such an estimation process is sketched. The observed behaviour from the actual phenomenon is compared to the predicted behaviour from the dynamical network model at some time points. If there is a significant difference (the aim is to make this difference minimal), the model's behaviour has to come closer to the observed behaviour by changing the values of the model's network characteristics.

From a more general perspective, for any type of dynamical model such (network) model characteristics are called *model parameters*. Various *parameter tuning methods* have been developed that specify how to quantify the deviation of a model's outcomes from real-world data and based on that in what way improved values of the model parameters can be determined that result in smaller deviations between model outcomes and real-world data.

In the current chapter this issue is discussed in some detail. In Sect. 19.2 it is discussed how in the many cases that direct measuring is hard or impossible, via requirements the values of parameters can be determined. In Sect. 19.3 an example of a temporal-causal network model is shown that is used in this chapter to illustrate the approach. In Sect. 19.4 parameter tuning based on exhaustive search and in Sect. 19.5 parameter tuning based on simulated annealing is discussed. Section 19.6 gives a brief overview of pros and cons of exhaustive search, simulated annealing and a few other approaches from the literature. In Sect. 19.7 it is shown how simulated annealing can be applied by using a specific component from the dedicated modeling environment and the optimiser Optimtool within MATLAB.

19.2 Determining Characteristics and the Use of Requirements

Without having precise knowledge of the specific characteristics of a real-world situation that is modelled, it will be difficult to obtain a model that really fits to the considered situation. A network model is specified by its network characteristics. The problem to find proper values representing characteristics for a given real-world situation basically can be addressed in two manners: by direct measuring of the characteristics in the situation that is modelled, or via requirements. These will be discussed in some more detail in the current section.

19.2.1 The Choice of Network Characteristics in a Network Model

In principle, all network characteristics of a given network model can be chosen for parameter tuning. However, first of all note that in an *adaptive* network model *only the non-adaptive* characteristics can be tuned as parameters; the adaptive characteristics are not constant and computed at each point in time within the model itself. Within a temporal-causal network model, in particular the following types of parameters occur that can be tuned (when they are not adaptive):

- Connection weights $\boldsymbol{\omega}_{X,Y}$ for states X and Y
- Speed factors $\boldsymbol{\eta}_Y$ for each state Y
- Combination function weights $\boldsymbol{\gamma}_{i,Y}$
- Parameters within specific combination functions, such as:
  - in a scaled sum combination function the scaling factor $\boldsymbol{\lambda}$
  - in a logistic sum combination function the parameters $\boldsymbol{\sigma}$ for steepness and $\boldsymbol{\tau}$ for excitability threshold
- For models with adaptive connections, for example:
  - for Hebbian learning the persistence parameter $\boldsymbol{\mu}_{\mathbf{W}_{X,Y}}$, where $\mathbf{W}_{X,Y}$ is a self-model state for adaptive connection weight $\boldsymbol{\omega}_{X,Y}$
  - for adaptive networks based on the homophily principle the tipping point $\boldsymbol{\tau}_{\mathbf{W}_{X,Y}}$ and speed factor $\boldsymbol{\eta}_{\mathbf{W}_{X,Y}}$ for connected X and Y, where $\mathbf{W}_{X,Y}$ is a self-model state for adaptive connection weight $\boldsymbol{\omega}_{X,Y}$.

Here, for example, in a specific real-world situation the connection weights may relate to the strengths of certain connections in someone's brain or to the strengths of certain connections in social interaction, and speed factors may relate to actual speed of processing the states. For a given situation it is not clear at forehand how values of such network characteristics have to be chosen. There are some indications or heuristics that can be kept in mind, to manage them during the modelling process:

- Connection weights 1 for maximal effect and lower between 0 and 1 for a smaller effect
- Connection weights between -1 and 0 for suppressing effects
- Speed factors can be chosen higher for internal, mental processes, and lower for body changes and execution of actions in the world
- For scaled sum combination function: choose the scaling factor $\boldsymbol{\lambda}$ equal to or at most the sum of the weights of the incoming connections
- For logistic sum combination functions
  - Choose steepness $\boldsymbol{\sigma}$ low (e.g., from 2 to 6) for gradual effects and high (e.g., 10 to 20 or higher) for more all-or-nothing types of effects
  - The excitability threshold $\boldsymbol{\tau}$ often has a relation with the number of incoming connections and their weights. For example, for activation values between 0 and 1, the sum of the incoming single impacts can never be more than the sum of these weights, so an excitability threshold $\boldsymbol{\tau}$ higher than that will not lead to substantial activation of a state; it usually has to be higher if there are more incoming connections

Such heuristics still will not make it easy to find values for network characteristics that adequately represent the specific characteristics of a given real-world situation. This basically can be addressed in two manners: in a direct manner by measuring of the network characteristics in the situation that is modelled, or in an indirect manner via requirements. These will be discussed in some more detail in Sects. 19.2.2 and 19.2.3, respectively.

19.2.2 Direct Measuring of Network Characteristics in a Real-World Situation

From a naïve point of view, the possibility to directly measure values of the network characteristics of a network model is the most attractive option. For example, if some physical process is to be modelled, according to some physical laws in which certain quantities (such as mass or volume) occur as characteristics, then the values of these quantities can be measured and used for these parameters. This may work in an idealised physical domain, but for human and social domains this may be less straight forward. Suppose a connection from one mental state X to another mental state Y is involved in the network model, then according to the current state of the art measuring the strength of this connection is quite difficult, if not impossible.

As another example, suppose in a network that models social interaction, a connection from person X to person Y occurs. How could the strength of this connection be measured? By the number of Whatsapp messages per minute? By the time duration of telephone calls? By the time duration of being at the same location? In the literature it is discussed how connection strength in networks describing social interaction relates to aspects such as interaction frequency, emotional intensity of content, and emotional support and closeness; e.g., (Gilbert and Karahalios 2009; Granovetter 1983; Marsden and Campbell 1990). In part of this literature, the relation between connection strength and aspects of actual interaction is used to formulate a measurable definition of connection strength. However, in other literature not a definitional but a causal relationship between such measurable aspects and connection strength is assumed; e.g., (Hove and Risen 2009; Pearce et al. 2015). So also direct measuring of connection strength in a network describing social interaction is not without problems.

As another alternative, sometimes questionnaires are used for measurements of characteristics in human or social domains, where persons can score their characteristics; for example, is some other person your partner, your best friend, just a friend or an acquaintance? Or, how close is this contact at a 5-point scale from close to not close? At first sight this may seem practical and adequate, as such a scoring might be used to generate numbers, and for numerical simulation numbers are needed. But this also has some problems. First of all, persons do not necessarily know their own characteristics, and if they believe they do know, there is no guarantee that these beliefs are correct. And secondly, a score from a questionnaire, or an aggregation of such scores, may provide a number, but this number is supposed to be measured according to some scale, and it may not be clear how this scale relates to the scale of a relevant network characteristic. There may be a nontrivial, unknown relation between such scales, perhaps at least a monotonic relation, but maybe not proportional or linear. So to adequately translate such scores into values for network characteristics can be a problem by itself.

19.2.3 Using Requirements to Find Characteristics of a Situation

Another way to find values for network characteristics of a network model is to identify and explicate what behavioural pattern the network model is expected to generate: expressed as requirements for the simulation outcomes of the model; see also (Treur 2016a), Ch. 13. Suppose such requirements have been identified, and they indeed describe what is expected from the model. Then a number of values of network characteristics can be tried alternatively until values are found such that the model shows the behaviour fulfilling the requirements. Usually this is already done intuitively by a modeller. However, for larger numbers of network characteristics the huge space of possibilities for a network model now turns into a huge search space. For models with many network characteristics it is easy to get lost in the large search space of all combinations of values of these network characteristics. So, eventually the question how to find proper values for the network characteristics may get an answer in the form of a search problem that is to be solved. The requirements used for this search problem can be of different types. In general the requirements can take the form of any temporal patterns expressed as dynamic properties as addressed in

(Treur 2016a), Ch. 13. Different combinations of values for network characteristics can be tried in order to find those combinations of values that lead to fulfilment of the requirements, or at least to approximate fulfilment. This process can be performed in a systematic manner, as exhaustive search, or some form of heuristic search. For heuristic search usually some measure is used to indicate how far from fulfilment the requirement is; this is often called an *error measure* or *error function*. The requirements can be of a very specific form when for some states of the model *empirical values* are available for some of the time points, and it is considered that the model is required to generate values at these time points equal or close to these empirical values.

19.2.4 Using Error Measures for Requirements Based on Data Points

The search methods discussed below in Sects. 19.4 and 19.5 assume requirements that indicate that the simulation values should be equal to an obtained set of data points representing the real-world situation and an error function to measure how far from fulfilment these requirements are. More specifically, it is assumed that a data set is available represented by values for certain states and time points, so basically a collection of values $V_{i,t}$ (i over some state numbers i_1, i_2, …, and t over some time points t_1, t_2, …). The requirement considered is that a trace generated by the model shows values for these states at these time points that are equal to the values indicated by the data points, or at least close to these values. As being equal is usually not feasible, the question becomes how to define this 'being close to' for multiple states and time points.

An error function expresses in an aggregated manner the overall deviation for all of the values of the data points in comparison to the simulation values for the considered states and time points. When there is no deviation for any of the states and time points, the error function will give the value 0, and if the deviation is small, the value of the error function will be close to 0. For a parameter tuning method the aim is to get the value of the error function below some small value (accuracy) or as close to 0 as possible. An error function usually is based on the absolute values of the differences $D_{i,t}$ (i over some state numbers i_1, i_2, …, and t over some time points t_1, t_2, …) between corresponding empirical and simulated values. Then a criterion could be that all of them should be at most a given small positive number D:

$$\left|D_{i,t}\right| \leq D$$

for all considered states X_i and all considered time points t. Here D is a small positive number, for example 0.05. Yet another way is to aggregate the deviations $D_{i,t}$ into one number, which is called an error function. One possible way of aggregation is to take the average absolute deviation:

Average absolute deviation error =

$$(|D_{i_1,t_1}| + |D_{i_1,t_2}| + \cdots + |D_{i_2,t_1}| + |D_{i_2,t_2}| + \cdots + \cdots)/N$$

or in short notation: $\Sigma_{i,t}|D_{i,t}|/N$, where N is the number of data points. However, an often used form of aggregation for an error function makes use of the sum of squares of the differences $D_{i,t}$: $\Sigma_{i,t}\ {D_{i,t}}^2$. The sum of squares of residuals as this is called as a basis for an error function is a generic concept that is used in applications in many disciplines to measure the deviation of a set of data points relative to a reference. The word residual refers to the difference between observed versus predicted values for the considered variable.

Minimizing the sum of squares of residual values is referred as a *least square method* (Moler 2004). The history of the least square method goes back to 1795, when Karl Friedrich Gauss has formulated it as a basic concept and found out that when it is assumed that for large numbers of data points measurements deviate from an ideal pattern according to a normal probability distribution, then a least square method provides an optimal approximation of the ideal pattern; see, for example, Strejc (1980). Note that this argument concerning a normal distribution of deviations assumes large data sets and is less valid for smaller data sets. To make the values obtained by calculating the sum of squares comparable with the actual differences $D_{i,t}$ it is useful to apply the square root of the avarage of the squares, obtaining what is called the Root Mean Square error RMSE which is also an often used variant:

$$\text{RMSE} = \sqrt{\Sigma_{i,t} D_{i,t}^2/N}$$

where N is the number of data points, which is often (but not necessarily) the product of the number of considered states and the number of considered time points. This makes it much easier and intuitive to verify whether the error makes sense, as the values of such an error function directly relate to the (linear) vertical differences that can be seen in the graph that compares simulated curves to empirical data points. For example, when all deviations $D_{i,t}$ are the same D, then this results in error RMSE = D.

In this way the RMSE-error function can be used to evaluate the quality of the selected values of the network characteristics in a given model in comparison to empirical data. Using any error function, there are different ways how to formulate a requirement. One most strict requirement would be that the error is 0. Although sometimes this may be relevant, in many practical situations such a requirement is too strict. Another option is to express in a requirement that the error is at most a given small value D, for example, 0.1:

$$\text{RMSE} \leq D$$

This can be used in a generate-and-test method that works by generating simulation traces under systematic variations of the settings of a model one by one and for each trace testing whether this requirement is fulfilled, until one is found that fulfills the requirement. When a model has many network characteristics, it may be difficult to generate traces for the many relevant variations of settings for the characteristics. In such cases often heuristic search methods are applied, such as the one (simulated annealing) that will be discussed in Sect. 19.5.

19.3 Description of an Example Model

The example described here is used to illustrate the idea of parameter tuning. This example network model for the generation of feelings follows Damasio's idea of as-if body loop (Damasio 1999; Damasio et al. 2000; Parvizi et al. 2006). A conceptual representation of the model is shown in Fig. 19.2; the states used in the model are summarized in Table 19.1. The model uses two inputs: stimulus s, and body state b, which may occur as a response to the stimulus. The stimulus s is associated to an emotional response b leading to a detectable body state ws_b (e.g., a face expression). In turn the effect ws_b serves as input by sensing it via ss_b.

World states ws_w (e.g., ws_s, and ws_b) affect sensor states ss_w (e.g., ss_s, and ss_b respectively). The sensor states lead to further internal processes according to the following causal sequence described by the body loop in Fig. 19.2. Moreover, the effect prediction loop or as-if body loop goes from preparation for bodily response

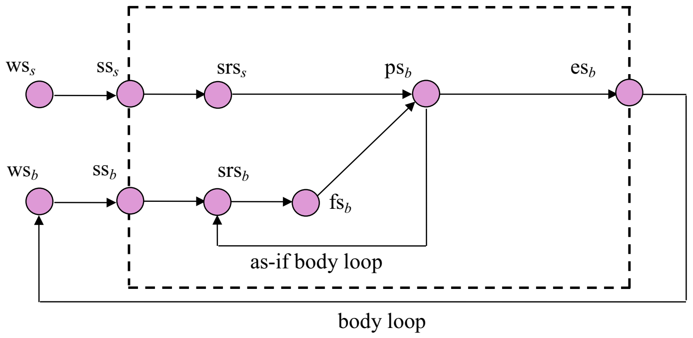


Fig. 19.2 Example model to illustrate parameter tuning

Table 19.1 States in the example model shown in Fig. 19.2

Notation	Description
ws_W	World state for W (W is a stimulus s, or body state b)
ss_W	Sensor state for W
srs_W	Sensory representation state for W
ps_b	Preparation state for emotional response b
fs_b	Feeling state for b
es_b	Execution state for response b

Table 19.2 Overview of the connections and their weights

To state	From state	Weight
ss_s	ws_s	ω_1
ws_b	es_b	ω_2
ss_b	ws_b	ω_3
srs_s	ss_s	ω_4
ps_a	srs_s fs_b	ω_5 ω_6
srs_b	ss_b ps_b	ω_7 ω_8
fs_b	srs_b	ω_9
es_b	ps_b	ω_{10}

to sensory representation of the bodily response to feeling the associated emotion (Damasio 1999; Damasio et al. 2000). The connections between state properties (the arrows in Fig. 19.2) have weights ω_k, as indicated in Table 19.2. In this model it is assumed that all weights are non-negative and between 0 and 1.

In the example simulations, for the states Y that are affected by only one state, the combination function $\mathbf{c}_Y(\ldots)$ is taken as the identity function $\mathbf{c}_Y(V) = \mathbf{id}(V) = V$, and for the other states ps_b and srs_b the combination function $\mathbf{c}_Y(\ldots)$ is the advanced logistic sum function $\mathbf{alogistic}_{\sigma,\tau}(\ldots)$.

$$\begin{aligned}\mathbf{c}_Y(V_1, \ldots, V_k) &= \mathbf{alogistic}_{\sigma,\tau}(V_1, \ldots, V_k) \\ &= \left(\frac{1}{1 + e^{-\sigma(V_1+\cdots+V_k-\tau)}} - \frac{1}{1 + e^{\sigma\tau}}\right)\left(1 + e^{-\sigma\tau}\right)\end{aligned}$$

Here $\boldsymbol{\tau}$ is the excitability threshold and $\boldsymbol{\sigma}$ is the steepness. For the speed factor $\boldsymbol{\eta}_X$ two values are used: $\boldsymbol{\eta}_{\mathrm{slow}}$ (slower) for external states X and $\boldsymbol{\eta}_{\mathrm{fast}}$ (faster) for internal states X: as sensor states and execution states need more time to change physically, the speed factor for these external states should be low compared to the ones for internal states. In the model the states ws_s, ws_b, ss_s, and ss_b are considered to be external. For the expected feeling when the stimulus has level 1, the data was chosen as shown in Fig. 19.3. For this case study these were generated by the model, after which according to a normal distribution some noise was added to make them look like empirical data.

The characteristics of the model considered for parameter tuning are 10 connection weight values ($\boldsymbol{\omega}_1$ to $\boldsymbol{\omega}_{10}$ in Table 19.2), 2 threshold values ($\boldsymbol{\tau}_{\mathrm{ps}_a}$ and $\boldsymbol{\tau}_{\mathrm{srs}_b}$), 2 steepness values ($\boldsymbol{\sigma}_{\mathrm{ps}_a}$ and $\boldsymbol{\sigma}_{\mathrm{srs}_b}$). It is assumed that only $\boldsymbol{\eta}_{\mathrm{slow}} = 0.5$ and $\boldsymbol{\eta}_{\mathrm{fast}} = 0.9$ have predefined values, and that the step size is $\Delta t = 0.25$. The values of the remaining 14 characteristics are to be determined by parameter tuning.

As an illustration simulated annealing will be applied to this example in Sect. 19.5. By applying some (relative) noise to the values of an example simulation according to a normal distribution, the data points shown in Fig. 19.3 were obtained. By parameter

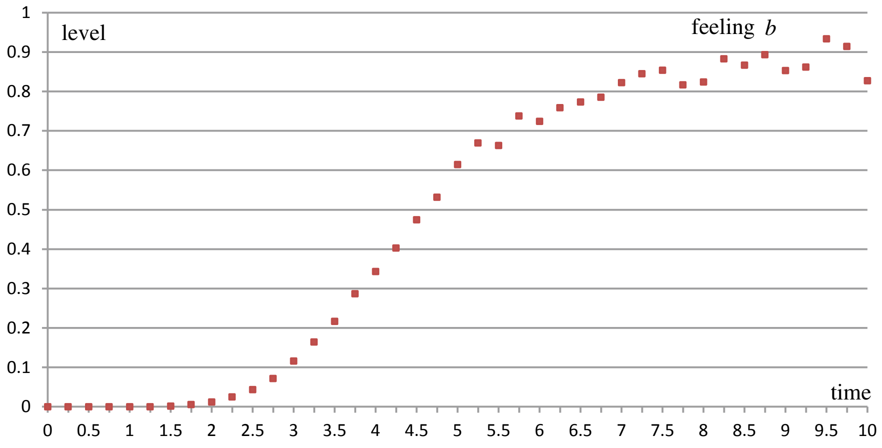


Fig. 19.3 The required behaviour of fs_b when the world state for stimulus ws_s is 1

tuning, values of the considered network characteristics will be determined that make that the simulated feeling level approximates the data points shown in Fig. 19.3. As error function, RMSE has been used, and as a stop criterion for the estimation process an error of 0.03.

Note that no unique solution can be expected. There may be different types of persons with different characteristics that still generate similar feeling levels. Different settings and initial values and different random choices on the fly of any parameter tuning method in principle will generate different solutions. It is often interesting to explore these different possibilities, and not blindly focus on only one option for which the error is just a few percentages lower.

19.4 Parameter Tuning by Exhaustive Search

Exhaustive search (also called brute-force search) is a quite elementary method for parameter tuning. It consists of

- Systematically enumerating all possible assignments of values to the network characteristics for the model, with a certain grain size or accuracy, for example in two or three decimals
- For each of these assignments run the model to generate a simulation trace
- For each generated simulation trace check whether (and to which extent) the generated simulation trace fulfils the requirements.

The option(s) that fulfill the requirements (or fulfill them best) are selected as suitable options for values of the parameters. If requirements are not fulfilled in a strict sense but only in an approximate sense, it is assumed that an error function is used, and the parameter values which show the lowest value for this error function can be chosen as the best outcome; here as error function, for example, the square root of the average of the squares of residuals may be used. For this case, example pseudo-code is shown in Box 19.1.

Box 19.1 Pseudo-code for

```
AP = first candidate assignment of values   % Initialization
least error = 10^10                         % Initialization
while (no terminate while)                  % For all possible parameter assignments
    run the model for AP                    % generate a simulation trace
    if (error(AP) <  least-error )          % if this assignment is better than the best one found yet
        best-set-of-parameter-values = AP   % the new set of parameters is the best until now
        least-error = error(AP)             % the error of the new set is the least one
    end if
    if assignment left
        AP = next-assignment(AP)            % go to the next value assignment
    else
        terminate-while
    end if
end while
```

Usually dynamical models have continuous parameters. To use exhaustive search for such models, the parameter values have to be to be assigned discrete values. The simplest way to do this is according to a uniform grain size, for example of 0.01; this grain size is a measure for the accuracy by which the search is performed. For example, suppose that the model has just one parameter P_1, which is continuous and it varies between 0 and 1. The aim is to find a value for P_1 which minimizes the difference between empirical data and model prediction. For example, for accuracy 0.1, parameter P_1 can be assigned discrete values 0.0, 0.1, 0.2, 0.3, … 0.9, 1.0, respectively. According to the exhaustive search method, the error (difference between empirical data and model prediction) for each of these values has to be determined, and the one which leads to the lowest error can be chosen.

As it is clear in Table 19.3, in this example, the error is minimal when P_1 is equal to 0.1. Thus, for this fictitious example, the exhaustive search method suggests to choose the value 0.1 for parameter P.

As another fictitious example, suppose that the model has two continuous parameters P_1, P_2, and values of both of them can be between 0 and 1. If the required accuracy for each of the parameters is taken 0.1, then for each one the values 0.0, 0.1, 0.2, 0.3, …, 0.9, 1.0 can be used, thus $11^2 = 121$ assignments of parameter

Table 19.3 Example error values for different parameter values

P_1	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Error	0.443	0.170	1.084	2.010	2.731	3.265	3.665	3.972	4.218	4.421	0.443

Table 19.4 Error of the prediction of the model with different set of parameters

$P_2\backslash P_1$	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0.0	0.114	0.083	0.064	0.056	0.061	0.077	0.105	0.145	0.197	0.261	0.337
0.1	0.058	0.022	0.003	0.002	0.02	0.058	0.116	0.193	0.292	0.412	0.553
0.2	0.12	0.086	0.077	0.096	0.144	0.223	0.333	0.476	0.655	0.87	1.124
0.3	0.299	0.278	0.294	0.35	0.449	0.594	0.789	1.037	1.342	1.708	2.139
0.4	0.596	0.603	0.661	0.776	0.953	1.199	1.52	1.923	2.416	3.008	3.706
0.5	1.011	1.062	1.184	1.385	1.675	2.065	2.566	3.191	3.954	4.871	5.958
0.6	1.543	1.659	1.871	2.192	2.638	3.226	3.975	4.907	6.045	7.417	9.052
0.7	2.192	2.398	2.731	3.212	3.865	4.717	5.798	7.143	8.792	10.788	13.181
0.8	2.96	3.282	3.772	4.46	5.382	6.578	8.095	9.987	12.316	15.152	18.574
0.9	3.845	4.314	5.002	5.952	7.216	8.853	10.933	13.537	16.759	20.707	25.508
1.0	4.847	5.498	6.431	7.707	9.399	11.592	14.387	17.904	22.284	27.688	34.309

values have to be evaluated for P_1 and P_2:

$$\begin{aligned}&\{(0.0, 0.0), (0.0, 0.1), (0.0, 0.2), \ldots,\\ &(0.1, 0.0), (0.1, 0.1), (0.1, 0.2), \ldots,\\ &\ldots\\ &(1.0, 0.8), (1.0, 0.9), (1.0, 1.0)\}\end{aligned}$$

So, for each of these assignments the error has to be determined (see Table 19.4) and then the set with least error chosen.

Table 19.4 illustrates the error for each of these 121 assignments of parameter values. The error is minimal when (P_1, P_2) is assigned values (0.3, 0.1). Thus, the exhaustive search method suggests 0.3 and 0.1 as best values for P_1 and P_2.

A problem with exhaustive search is that for more parameters and smaller grain size it becomes computationally more complex. Therefore, exhaustive search is typically used as the best method when the problem size (e.g., the number of parameters and required accuracy) is limited; in other cases this method may easily become too inefficient.

19.5 Parameter Tuning by Simulated Annealing

In a purely random approach, the parameter values are randomly changed. Like exhaustive search such a random method is complete, in the sense that it will always find a global minimum in the end, but it can be quite inefficient. Simulated annealing is a slightly more sophisticated method which uses randomness but adds over time more and more focusing to the randomness. This makes it more efficient but in principle does not find the global optimum anymore. The most important advantage of this method (in comparison to some other methods) is that simulated annealing can still pull the tuning process out of local minima.

Simulated annealing is inspired by physical annealing in metallurgy, where physical substances are heated and melted, and then gradually cooled down until some solid state is reached. In this process, the goal is reaching a state in which the substance has a minimum of energy. In metallurgy, this goal will be attained if the substance is cooled down in a sufficiently slow manner. The notion of slow cooling down in physical annealing is implemented in the simulated annealing method as a slow decrease in the probability of accepting worse solutions while it explores the search space. This property is controlled by a variable called Temperature (T). Accepting a worse assignment of parameter values (which leads to higher value of error) is a fundamental property of simulated annealing because it allows for a more extensive search for the optimal solution.

Simulated annealing starts from a random assignment of parameter values AP. At each step, the method considers a neighbouring assignment of parameter values AP' of the current set AP, and probabilistically decides between moving to set AP' or staying in AP. These probabilities ultimately lead the system to move to better sets of parameter values (with lower error). Inspired from physical annealing, in this method there is a variable called temperature (indicated by T), which has a value decreasing (cooling) during the process. The temperature controls the probability of doing ‘downhill’ actions during the process. When the value of this parameter is high (at the beginning) the probability of accepting a new set of parameter which leads to higher error is high. During the progress of the method, this value will decrease, and when the value of this variable is very low (at the end), the probability of accepting such a new set is almost zero, which means that the process is now focusing on that part of the search space only.

Because of the possible jumps to worse solutions when the temperature is still high, simulated annealing can pull out of local minima and may be able to find the globally most optimal point, especially if the temperature is cooled down in a sufficiently slow manner.

More specifically, it starts from a random set of parameter values AP^0 and generates a succession of parameter values sets AP^1, AP^2, … in order to decrease the error. New candidate sets are generated around the current set of parameter values by slightly changing these values in a random way for each parameter. For each iteration i the new values are uniformly distributed in intervals centered around AP^i. If the new assignment of parameter values has a decreased the error level error(AP^{i+1}), in comparison to the error level error(AP^{i+1}) of the previous assignment, it will be accepted (i.e., when $\Delta E = \text{error}(AP^{i+1}) - \text{error}(AP^i) \leq 0$). Otherwise, the new set will be accepted with probability $e^{-\Delta E/T}$, which depends on the temperature and the difference between previous error and the new one. So, a new assignment of parameter values which generates a higher level of error, will be accepted with probability of $e^{-\Delta E/T}$. This probability is only dependent on ΔE and T. If ΔE is very small (the new assignment of parameter values increases the error level only a bit), or if the temperature is very high, then the new assignment of parameter values will be accepted with a high probability. This probability will decrease by decreasing the temperature or increasing ΔE.

Figure 19.4 shows $e^{-\Delta E/T}$ as a function of ΔE when $T = 1$. Figure 19.5 shows $e^{-\Delta E/T}$ as a function of T when $\Delta E = 2$.

As it can be seen, the value of the function $e^{-\Delta E/T}$ is close to 1 when ΔE is very small or T is large. On the other hand, it is less when ΔE is large and T is small. The simulated annealing method starts at a user defined temperature T_0 and the temperature will be decreased in each iteration. The process is terminated when the temperature is so low that no more significant improvement can be reached. See Box 19.2 for pseudocode. So, the following happens:

- If the move improves the situation (decreases the error), it is always accepted.
- Otherwise, the algorithm accepts the move to a higher error with some probability less than 1.

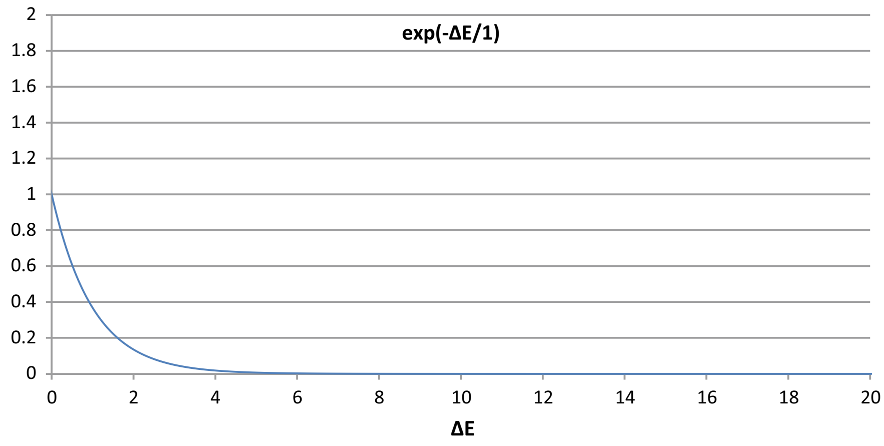


Fig. 19.4 The values of $\exp(-\Delta E/T)$ depending on ΔE for $T = 1$

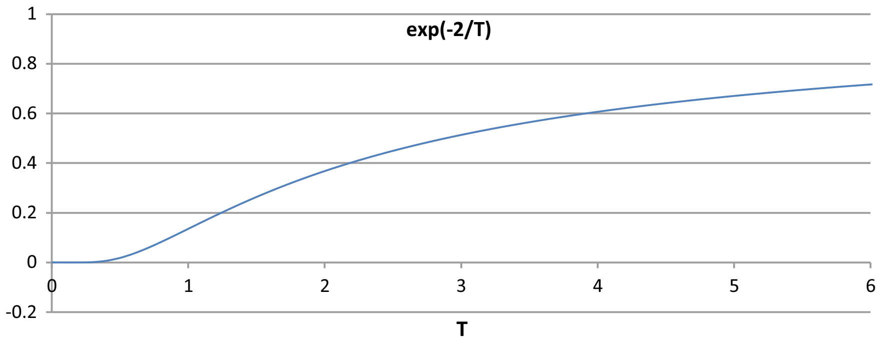


Fig. 19.5 The values of $\exp(-\Delta E/T)$ depending on T for $\Delta E = 2$

- The probability decreases exponentially with how bad the move is: the amount ΔE by which the error is increased.
- The probability also decreases as the temperature T goes down: bad moves are more likely to be allowed at the start when T is high, and they become more unlikely as T decreases.
- If the temperature T decreases slowly enough, the method will find the best set of parameters with probability approximating 1.

Box 19.2 Pseudo-code for simulated annealing

```
Set λ =  cooling rate of temperature          % a value between 0.9 and 1.0, like  0.995
Set α =  neighbourhood limitation             % a value like 0.1
Set initial T                                 % Initialization
Set initial AP randomly                       % Initialization
Run the model for AP to generate a trace      % Generate simulation trace for AP
Set E= error (AP) for this trace              % Calculate the error for assignment AP
While (T ≥ Tmin)
    Generate AP' from AP by for each parameter value P in AP
        set P' = P + α * (rand() - 0.5)       % Choose a neighbouring assignment AP'
    Run the model for AP' to obtain a trace   % Generate simulation trace for AP'
    Set E' = error(AP') for this trace        % Calculate the error for assignment AP'
    ΔE= E' – E
    If (ΔE<  0 )                              % If the new assignment AP'  reduces the error
        AP = AP'                              % change assignment to one with lower error
        E = E'                                % update the error value
    else                                      % if the assignment AP' does not reduce the error
        if   random() > exp(-ΔE / T)          % do the next step with the probability exp(-ΔE/ T )
            AP = AP'                          % change assignment to one with higher error
            E = E'                            % Update the error value
        end if
    end if
    T = T * λ                                 % Decrease the temperature
end while

random() = a function that generates a random value between 0 and 1
```

As mentioned, the temperature T is decreased during the progress; the cooling schedule defined by parameter λ is very important for the performance of simulated annealing. There is a trade-off between the quality of the final solution and the execution time, the latter being sensitive to the speed of the temperature decrease. Here, it is done by multiplying T in each iteration with λ which is a number <1 (and usually >0.95). If a very high initial temperature T is chosen, there will be a waste of computational resources. In the case of low initial T, the process could get caught in assignments of parameter values which are not the best ones (local minima). It is hard to establish a general rule for determining the ideal initial temperature.

The simulated annealing method has been implemented for the example described in Sect. 19.3. The upper graph in Fig. 19.6 shows the deviations for the found solution. The lower graph in Fig. 19.6 shows the error for the different iterations.

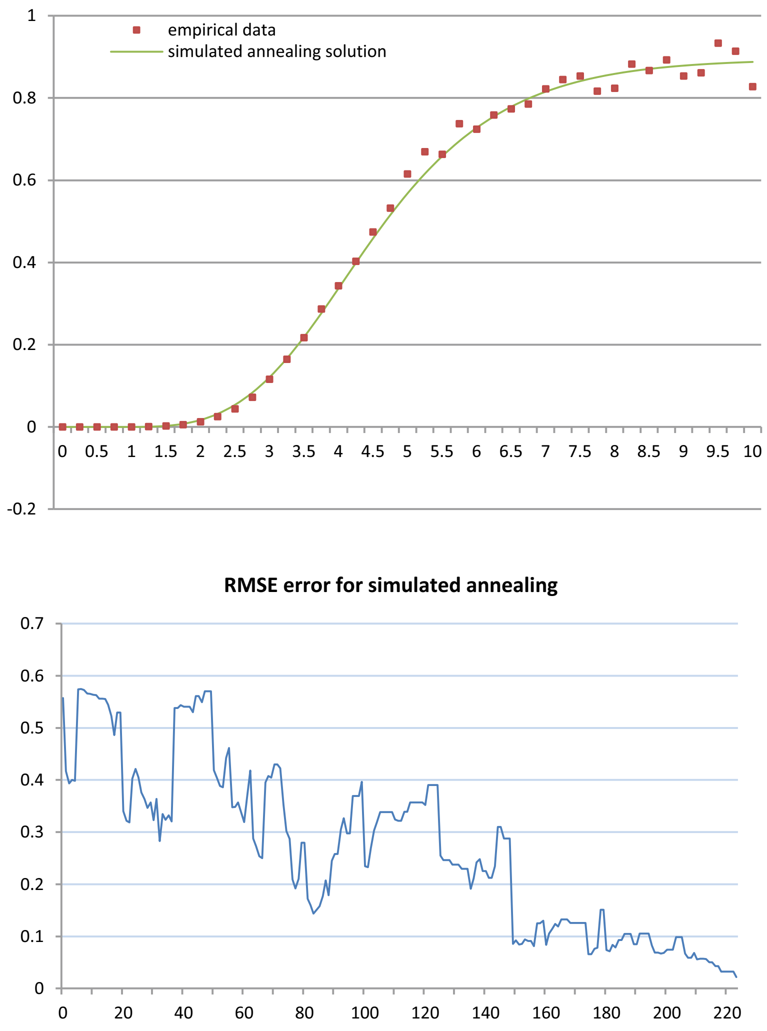


Fig. 19.6 The deviations and error for the different iterations in simulated annealing

19.6 Pros and Cons of Different Parameter Tuning Methods

In (Treur 2016a), Ch. 14 two other parameter tuning methods (gradient descent and random gradient descent) are described as well and also for them it is illustrated how they work for the example in Sect. 19.3. In the current section, first different solutions obtained by different methods are shown (in Table 19.5) for the example model of Sect. 19.3.

As should be expected, different parameter tuning methods and settings found different solutions for the example model. Below the parameter values are shown as

Table 19.5 Three different solutions found by three different parameter tuning methods

	Gradient descent	Random gradient descent	Simulated annealing
$\boldsymbol{\omega}_{\mathrm{es}_b,\mathrm{ws}_b}$	0.64	0.92	0.59
$\boldsymbol{\omega}_{\mathrm{ws}_s,\mathrm{ss}_s}$	0.82	1.00	0.52
$\boldsymbol{\omega}_{\mathrm{ws}_b,\mathrm{ss}_b}$	0.71	0.90	0.97
$\boldsymbol{\omega}_{\mathrm{ss}_s,\mathrm{srs}_s}$	0.85	1.00	0.55
$\boldsymbol{\sigma}_{\mathrm{srs}_b}$	2.84	2.79	6.94
$\boldsymbol{\tau}_{\mathrm{srs}_b}$	0.01	0.10	0.01
$\boldsymbol{\omega}_{\mathrm{ss}_b,\mathrm{srs}_b}$	0.69	0.78	0.73
$\boldsymbol{\omega}_{\mathrm{ps}_b,\mathrm{srs}_b}$	1.00	1.00	1.00
$\boldsymbol{\sigma}_{\mathrm{ps}_b}$	6.58	6.59	5.00
$\boldsymbol{\tau}_{\mathrm{ps}_b}$	0.10	0.24	0.10
$\boldsymbol{\omega}_{\mathrm{fs}_b,\mathrm{ps}_b}$	0.89	1.00	0.97
$\boldsymbol{\omega}_{\mathrm{srs}_s,\mathrm{ps}_b}$	0.88	1.00	1.00
$\boldsymbol{\omega}_{\mathrm{srs}_b,\mathrm{fs}_b}$	1.00	1.00	0.90
$\boldsymbol{\omega}_{\mathrm{ps}_b,\mathrm{es}_b}$	0.66	0.62	0.50

found by three different parameter tuning methods; note that all of these solutions had error 0.03 or just below that value.

This illustrates that usually there is not one unique best solution (for example, related to a global optimum) that is the only relevant solution, but multiple solutions (related to local optima) are possible and relevant. It may depend on the context which of these solutions are most relevant. In practice, to simply go for some unique solution related to a global optimum, as sometimes may be suggested, will often not be recommendable. Instead, better insight will be obtained when an overview is found of different solutions relating to local optima with their respective errors, especially when these errors are not that different, as in the above case.

Next, a summarized overview of pros and cons of the different methods can be found in Table 19.6; see also (Treur 2016a), Ch. 14.

19.7 Applying Parameter Tuning by the Modeling Environment

Within the dedicated modeling environment used for the network-oriented modeling for self-modeling adaptive networks approach presented in (Treur 2020) two tuning templates are available, one for nonadaptive network models and one for adaptive network models; they can be found here:

https://www.researchgate.net/project/Network-Oriented-Modeling-Software

Table 19.6 Overview of pros and cons for a number of parameter tuning methods

Advantages	Disadvantages
Simulated annealing	
• Simulated annealing is effective in finding a good assignment of parameter values for models with a huge numbers of parameters • Although the final point is not deterministically guaranteed to be the best one, if the temperature decreases in an appropriate pace, it will find a very good answer	• In comparison to gradient descent or random gradient descent, it is slow. This method needs more iterations to converge to an optimal answer
Exhaustive search	
• Exhaustive search is very simple to implement • It will always find the best set of parameters (global optimum) if all the possible parameter value combinations are explored • Exhaustive search is useful as 'baseline' method when benchmarking other algorithms	• The main disadvantage of the exhaustive search is that for many real-world problems the set of candidate value assignments is prohibitively large. Its computational cost is proportional to the number of value assignments, which in many practical problems tends to grow very quickly as the size of the problem increases, in particular when the number of parameters is large and/or a high accuracy is needed
Gradient descent	
• Gradient descent is relatively simple • It is a fast method; in a few iterations, it can find a local minimum • With certain assumptions, convergence to a local minimum can be guaranteed	• The outcome of gradient descent method may depend on the initial point • This method can get stuck in local minima • Many calculations have to be done in each iteration • There are situations in which gradient descent can be slow and inefficient. To overcome such problems, a number of variations on gradient descent have been developed, such as conjugate gradient descent (Chong and Zak 2013)
Random gradient descent	
• Random gradient descent is relatively simple • Not many calculations have to be done in each iteration; no sensitivities have to be determined • With certain assumptions, convergence to a local minimum can be guaranteed	• Many iterations may be needed before it reaches a small error • The outcome of gradient descent method may depend on the initial point • This method can get stuck in local minima

In this section the use of them for validation by parameter tuning is discussed.

19.7.1 Basic Elements Needed for Parameter Tuning

The following basic elements are needed for parameter tuning of network models.

19.7.1.1 Empirically Justified Data Points

Empirical data for one or more of the states in the model is assumed to be gathered from real world observations and experiments, or based on requirements formulated for the model, which usually also are based on empirical literature. Usually this is only for a small subset of the set of all states in the model, as for many of the states it may be difficult to get data. And the data usually also will only concern a small subset of the time points that the model uses. Various case studies can be used to develop empirical data, and it can play an important part to get models closer to reality. Some of the options to obtain data points are:

- Different scientific methods like fMRI, body state measuring (heart rate, skin conductance,…) can be used.
- Data can be obtained from social media, such as emotion levels extracted from sentiment analysis applied to posts.
- If numerical data are not available also qualitative empirical information can be useful to get data points. For example, if qualitative information indicates that at some point in time some activation level is low or high, you might map such a linguistic score, for example, on numbers 0.2 or 0.3, or 0.7 or 0.8, respectively (assuming the use of the [0, 1] interval).

These data are assumed to relate to one or some of the states of the model, and to some of the time points; thus you will have:

- The **time points** for which you can obtain data.
  These time points are usually given to the simulation environment in the form of a list or row (a one-dimensional matrix) of some length N. For example,

  $$\text{Timepoints} = [11.5\ \ 16]$$

  of length $N = 2$ indicates that data is available (only) for the indicated time points 11.5 and 16.
- The specific **states** for which you can obtain data
  These selected states are given to the software environment in the form of a list or row (a one-dimensional matrix) of some length M. For example,

  $$\text{Stateselection} = [X_2\ \ X_4\ \ X_5]$$

  of length $M = 3$ expresses that the data relate to the three indicated states X_2, X_4, and X_5.

- The **data points** themselves

  They have the form of a 2-D matrix with dimensions $M \times N$, where M (the number of rows) refers to the length of the state selection list above, and N (the number of columns) refers to the length the time points list above. For example, here the data will have the form of a 3×2 matrix.

19.7.1.2 Choice of Network Characteristics Serving as Parameters to Be Tuned

You also have to indicate the *network characteristics* from the value matrices or initial values list which you want to use as parameters to be tuned. For example, a choice to address tuning of (some) speed factors and (some) connection weights. Note that adaptive characteristics cannot be chosen for tuning as they change within a simulation.

19.7.1.3 Choice of Parameter Intervals

For each of the chosen parameters you have to specify the interval indicating the *lower* and *upper bound* of the range for the considered values for that parameter. For example, the interval [0, 1] or a smaller or bigger interval.

19.7.2 Preparation for the Tuning Process

Using one of the tuning templates, the actual preparation goes as follows.

19.7.2.1 Specifying the Table of Data Points

At forehand specify the information on the data points used by a data matrix and two lists (see also Sect. 19.7.1); then copy them to MATLAB:

(a) The time points list (5 elements here)
(b) The state selection list (3 elements here)
(c) The table of data points (a 3×5 matrix here).

Pasting them between the proper [] in MATLAB results in (here the X is left out from the state selection list):

```
timepoints=[2 5.9 14.3 21.1 29.2];
stateselection=[1 6 12];
empirical_data = [0.90 0.82 0.72 0.67 0.63
0.54 0.52 0.49 0.47 0.46
0.03 0.13 0.24 0.29 0.33
];
```

Here the rows correspond to the state selection and the columns to the selected time points. Recall that such data points can be based on, e.g.:

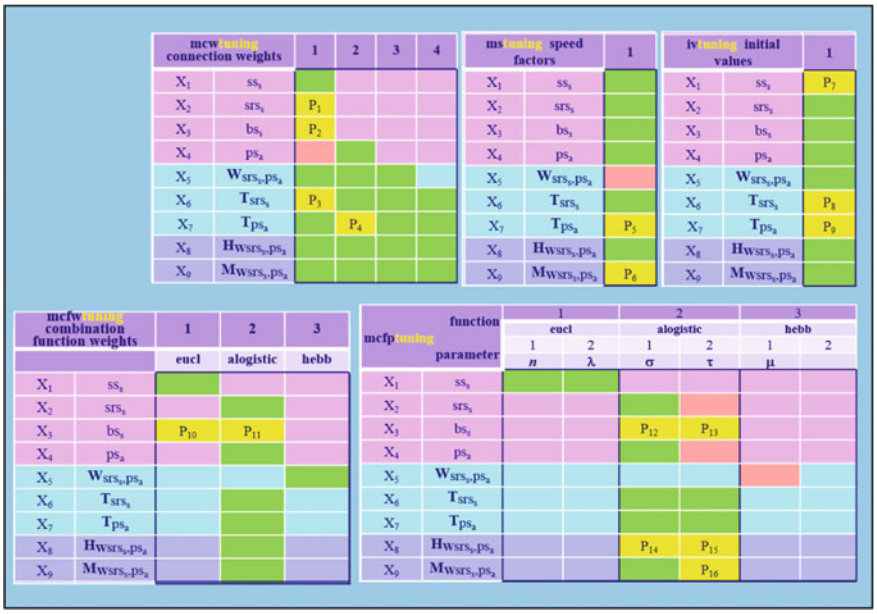
- A questionnaire or scores of an individual at a certain scale
- Posts in Social Media
- Requirements on the expected outcome acquired from qualitative empirical literature.

19.7.2.2 Specifying the Tuning Matrices

Next, specify the tuning matrices **mcwtuning**, **mstuning**, **mcfwtuning**, and **mcfp-tuning** and also **ivtuning**.

- All values for network characteristics in the value matrices **mcwv**, **mcfwv**, **mcfpv**, **msv** and in **iv** can be candidates for tuning, but not values in **mb** and the adaptation matrices **mcwa**, **mcfwa**, **mcfpa**, **ms**.
- Specify which of these characteristics are picked to be tuned and associate them to numbered parameters P(1), P(2),…
- This association is specified in **mcwtuning**, **mstuning**, **mcfwtuning**, and **mcfp-tuning** and also **ivtunin** by writing in these matrices P_1, P_2, … (for P(1), P(2), …) in the cells corresponding to the characteristics you picked

This will result, for example, in the tuning matrices shown in Fig. 19.7.



mcwtuning connection weights		1	2	3	4
X_1	ss_s	🟩			
X_2	srs_s	P_1			
X_3	bs_s	P_2			
X_4	ps_a	🟥	🟩		
X_5	W_{srs_s,ps_a}	🟩	🟩	🟩	
X_6	T_{srs_s}	P_3	🟩	🟩	🟩
X_7	T_{ps_a}	🟩	P_4	🟩	🟩
X_8	$H_{W_{srs_s,ps_a}}$	🟩	🟩	🟩	🟩
X_9	$M_{W_{srs_s,ps_a}}$	🟩	🟩	🟩	🟩

mstuning speed factors		1
X_1	ss_s	🟩
X_2	srs_s	🟩
X_3	bs_s	🟩
X_4	ps_a	🟩
X_5	W_{srs_s,ps_a}	🟥
X_6	T_{srs_s}	🟩
X_7	T_{ps_a}	P_5
X_8	$H_{W_{srs_s,ps_a}}$	🟩
X_9	$M_{W_{srs_s,ps_a}}$	P_6

ivtuning initial values		1
X_1	ss_s	P_7
X_2	srs_s	🟩
X_3	bs_s	🟩
X_4	ps_a	🟩
X_5	W_{srs_s,ps_a}	🟩
X_6	T_{srs_s}	P_8
X_7	T_{ps_a}	P_9
X_8	$H_{W_{srs_s,ps_a}}$	🟩
X_9	$M_{W_{srs_s,ps_a}}$	🟩

mcfwtuning combination function weights		1	2	3
		eucl	alogistic	hebb
X_1	ss_s	🟩		
X_2	srs_s		🟩	
X_3	bs_s	P_{10}	P_{11}	
X_4	ps_a		🟩	
X_5	W_{srs_s,ps_a}			🟩
X_6	T_{srs_s}		🟩	
X_7	T_{ps_a}		🟩	
X_8	$H_{W_{srs_s,ps_a}}$		🟩	
X_9	$M_{W_{srs_s,ps_a}}$		🟩	

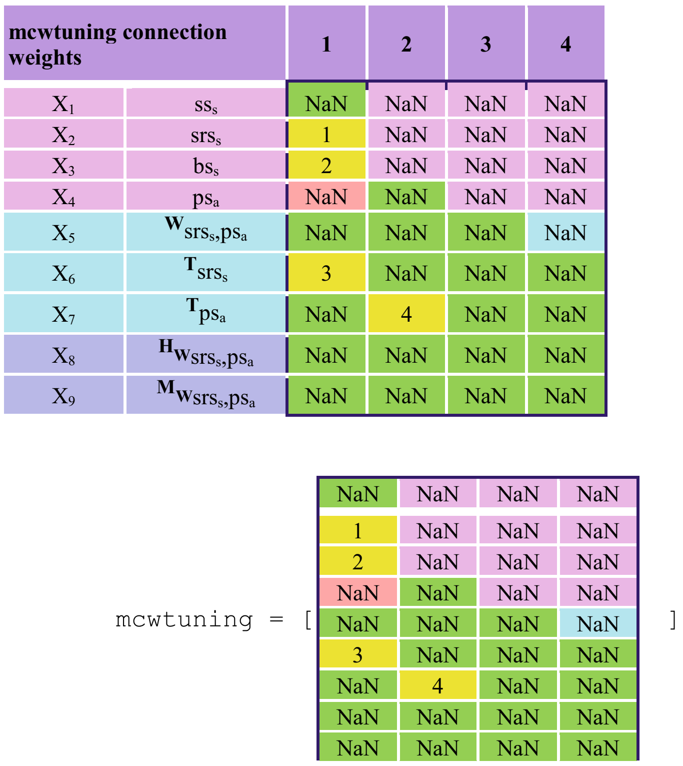
mcfptuning	function	1		2		3	
		eucl		alogistic		hebb	
	parameter	1	2	1	2	1	2
		n	λ	σ	τ	μ	
X_1	ss_s	🟩	🟩				
X_2	srs_s			🟩	🟥		
X_3	bs_s			P_{12}	P_{13}		
X_4	ps_a			🟩	🟥		
X_5	W_{srs_s,ps_a}					🟥	
X_6	T_{srs_s}			🟩	🟩		
X_7	T_{ps_a}			🟩	🟩		
X_8	$H_{W_{srs_s,ps_a}}$			P_{14}	P_{15}		
X_9	$M_{W_{srs_s,ps_a}}$			🟩	P_{16}		

Fig. 19.7 Examples of tuning matrices

19.7.2.3 Copying the Tuning Matrices into the Software Template

The tuning matrices obtained in Sect. 19.7.2.2 can be copied into the tuning template after leaving out the P and fill the empty cells with NaN; for example, for **mcwtuning**. See Fig. 19.8. This paste results in the following Tuning Matrix in MATLAB

```
mcwtuning = [NaN NaN NaN NaN
1 NaN NaN NaN
2 NaN NaN NaN
NaN NaN NaN NaN
NaN NaN NaN NaN
3 NaN NaN NaN
NaN NaN NaN NaN
NaN NaN NaN NaN
NaN NaN NaN NaN
];
```



mcwtuning connection weights		**1**	**2**	**3**	**4**
X_1	ss_s	NaN	NaN	NaN	NaN
X_2	srs_s	1	NaN	NaN	NaN
X_3	bs_s	2	NaN	NaN	NaN
X_4	ps_a	NaN	NaN	NaN	NaN
X_5	$\mathbf{W}_{srs_s,ps_a}$	NaN	NaN	NaN	NaN
X_6	$\mathbf{T}_{srs_s}$	3	NaN	NaN	NaN
X_7	$\mathbf{T}_{ps_a}$	NaN	4	NaN	NaN
X_8	$\mathbf{H}_{\mathbf{W}_{srs_s,ps_a}}$	NaN	NaN	NaN	NaN
X_9	$\mathbf{M}_{\mathbf{W}_{srs_s,ps_a}}$	NaN	NaN	NaN	NaN

mcwtuning = [

NaN	NaN	NaN	NaN
1	NaN	NaN	NaN
2	NaN	NaN	NaN
NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN
3	NaN	NaN	NaN
NaN	4	NaN	NaN
NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN

]

Fig. 19.8 Tuning matrix prepared for copying to MATLAB (upper) and pasting it in the tuning template: the right hand part is pasted between the [] of `mcwtuning` in MATLAB

Important hint:

- a Tuning Template does include all values matrices (and adaptation matrices) but cannot directly be used 'by hand' to generate a simulation by itself; it can only be used automatically in an indirect manner by running the optimiser Optimtool (via function calls within Optimtool).

19.7.3 Running the Tuning Process

After the above preparations, the parameter tuning process can be started. This goes as follows.

19.7.3.1 Enter Input in Optimtool

To run the optimization tool write 'optimtool' in the command window of MATLAB and press Enter; this opens Optimtool. Under Solver select the Simulated Annealing algorithm as shown in Fig. 19.9. You have to provide options in the window (see Fig. 19.10) such as

(a) starting point, and intervals for the parameter values
(b) lower and
(c) upper bound for values of the parameters

For example:

- Objective function = @NOMEtuningnonadaptivev02
- Start point: which parameter values you want to start with
- Lower Bounds = [0 0 0 0 0 0 0 0 0 0 0 0]
- Upper Bounds [1 1 1 1 1 1 1 1 1 1 1 1].

Note that depending on the parameters chosen, the 0 and 1 values can be different, for example for steepness parameters they might be chosen 3 (lower bound) and 15 (upper bound) instead. Make sure your parameters setting look like as shown in the image below.

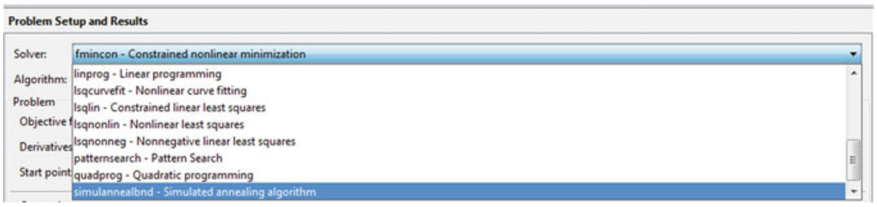
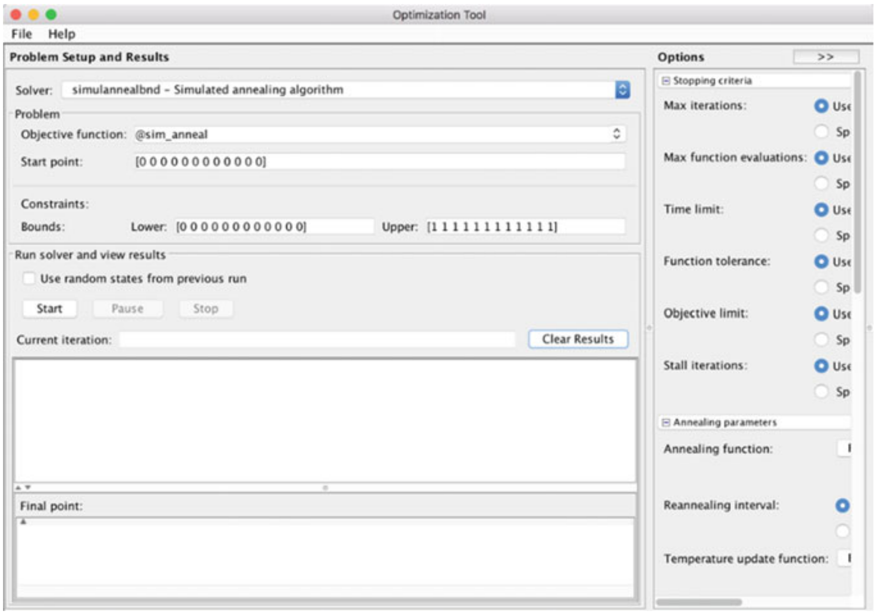

Problem Setup and Results
Solver:
fmincon - Constrained nonlinear minimization
Algorithm:
Problem
Objective f
Derivatives
Start point
linprog - Linear programming
lsqcurvefit - Nonlinear curve fitting
lsqlin - Constrained linear least squares
lsqnonlin - Nonlinear least squares
lsqnonneg - Nonnegative linear least squares
patternsearch - Pattern Search
quadprog - Quadratic programming
simulannealbnd - Simulated annealing algorithm

Fig. 19.9 Optimtool interface for solver and algorithm selection


Optimization Tool

Problem Setup and Results

Solver: simulannealbnd - Simulated annealing algorithm

Problem

Objective function: @sim_anneal

Start point: [0 0 0 0 0 0 0 0 0 0 0 0]

Constraints:

Bounds: Lower: [0 0 0 0 0 0 0 0 0 0 0 0] Upper: [1 1 1 1 1 1 1 1 1 1 1 1]

Run solver and view results

Use random states from previous run

Current iteration:

Final point:

Options

Stopping criteria

Max iterations:

Max function evaluations:

Time limit:

Function tolerance:

Objective limit:

Stall iterations:

Annealing parameters

Annealing function:

Reannealing interval:

Temperature update function:

Fig. 19.10 Optimtool interface for start point, lower bounds and upper bounds for the parameters used

Important hint:

- Always make the name of the file for the tuning template and the name of the function in the first line within that file the same; MATLAB wants the file names of functions to be equal to the function names

19.7.3.2 Running Optimtool

After the above preparations, just press Start

Important hint:

- If you already did a run with a number of iterations and you changed something, for example, the number of parameters addressed, then before starting again better apply

  ```
  >> clear P
  ```

  in the Command Window; otherwise old and new values and format for P may be confused by MATLAB. Especially do this when you get a `concatenation error` message.

19.7.4 How It Works

This section explains some remaining parts of the tuning templates.

19.7.4.1 Integrating the Parameter Values in the Values Matrices

After receiving the values of P(1), P(2), … from the optimizer Optimtool, based on the tuning matrices, the value matrices are updated by the software by inserting these values of P(1), P(2), … in the indicated cells mcwv, msv, mcfwv and mcfpv; for example:

```
%%%%Updating mcwv
[rows,cols] = size(mcwtuning);
for i = 1:rows
    for j = 1:cols
        if (~isnan(mcwtuning(i,j)))
            index = mcwtuning(i,j);
            mcwv(i,j) = P(index);
        end
    end
end
```

19.7.4.2 Determining the RMSE Error

To determine the RMSE, first the right portion of the model output is extracted to get specific data from the simulation corresponding to the chosen data points:

```
for j=1:length(stateselection)
  for i=1:length(timepoints)
  Y(j,i)=X(stateselection(j),1+round(timepoints(i)/dt));
  end
  model_data = Y;
end
```

Then RMSE is calculated as

- the sum of the squares of the deviations for all data points
- dividing this sum by the product of row and column length of the data table
- applying the square root:

```
[e_row,e_col] = size(empirical_data);
RMSE    =    sqrt(nansum(nansum((model_data    -    empir-
ical_data).^2))/(e_col * e_row))
```

The resulting value for this RMSE goes to the optimiser Optimtool, so that it can analyse the result and start a new iteration.

19.8 Discussion

This chapter is partly based on work with Fakhra Jabeen, Nimat Ullah, Amin Tabatabaei and Dilhan Thilakarathne. For example, the data and figures used were generated by Amin and Dilhan. In this chapter the focus was on Simulated Annealing as parameter tuning method. Some other methods can be found in (Treur 2016a), Ch. 14; sometimes variants of these methods or other methods are used; for example, see (Van den Bos 2007; Chong and Zak 2013; Thilakarathne 2015).

These methods use a specific simple type of requirements concerning equality to some data points. In general more complex temporal patterns can be relevant as requirements to be used, such as described as dynamic properties in a temporal language in (Treur 2016a), Ch. 13. In (Treur 2016b) an approach is introduced in which an error function is defined for any such a more complex requirement expressed as a dynamic property in the temporal language as discussed in (Treur 2016a), Ch. 13. This error function is based on the notion of 'approximate satisfaction' introduced in the mentioned reference, which provides a measure for how close to satisfaction an arbitrary temporal requirement is. It generalises the error function based on least squares of residuals when simulated values are compared to empirical values to the case of requirements for arbitrary temporal patterns.

References

Chong, E.K.P., Zak, S.H.: An Introduction to Optimization. Wiley (2013)

Damasio, A.: The Feeling of What Happens: Body, Emotion and the Making of Consciousness. Harcourt Brace (1999)

Damasio, A., Grabowski, T.J., Bechara, A., Damasio, H., Ponto, L.L.B., Parvizi, J., et al.: Subcortical and cortical brain activity during the feeling of self-generated emotions. Nat. Neurosci. **3**, 1049–1056 (2000)

Gilbert, E., Karahalios, K.: Predicting tie strength with social media. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI'09, pp. 211–220 (2009)

Granovetter, M.: The strength of weak ties: a network theory revisited. Sociol Theory **1**, 201–233 (1983)

Hove, M.J., Risen, J.L.: It's all in the timing: interpersonal synchrony increases affiliation. Soc. Cognit. **27**, 949–960 (2009). https://doi.org/10.1521/soco.2009.27.6.949

Marsden, P.V., Campbell, K.E.: Measuring tie strength. Soc. Forces **63**, 482–501 (1990)

Moler, C.: Least squares. In: Numerical Computing with MATLAB, Chapter 4. Society for Industrial and Applied Mathematics (2004)

Parvizi, J., van Hoesen, G.W., Buckwalter, J., Damasio, A.: Neural connections of the posteromedial cortex in the macaque: implications for the understanding of the neural basis of consciousness. Proc. Natl. Acad. Sci. **103**(5), 1563–1568 (2006)

Pearce, E., Launay, J., Dunbar, R.I.M.: The ice-breaker effect: singing together mediates fast social bonding. Roy. Soc. Open Sci. (2015). https://doi.org/10.1098/rsos.150221

Strejc, V.: Least squares parameter estimation. Automatica **16**(5), 535–550 (1980)

Thilakarathne, D.J.: A parameter estimation method for dynamic computational cognitive models. In: Proceedings of the 6th Annual International Conference on Biologically Inspired Cognitive Architectures, BICA'15, Procedia Computer Science, vol. 71, pp. 133–142 (2015)

Treur, J.: Network-Oriented Modeling: Addressing Complexity of Cognitive Affective and Social Interactions. Springer Publishers, Cham (2016a)

Treur, J.: Using automated approximate satisfaction in parameter search for dynamic agent models. In: Proceedings of the 14th European Conference on Multi-Agent Systems, EUMAS'16. Lecture Notes in AI, vol. 10207, pp. 230–247. Springer Publishers (2016b)

Treur, J.: Network-Oriented Modeling for Adaptive Networks: Designing Higher-Order Adaptive Biological, Mental and Social Network Models. Springer Nature Publishers, Cham (2020)

Van den Bos, A.: Parameter Estimation for Scientists and Engineers. Wiley Interscience, Hoboken, N.J (2007)