For office use only	Team Control Number	For office use only
T1 _____	**2017061**	F1 _____
T2 _____		F2 _____
T3 _____		F3 _____
T4 _____		F4 _____

2017

The International Mathematical Modeling Challenge (IM^{2}C) Summary Sheet

(Your team's summary should be included as the first page of your electronic submission.)

1 Summary

Trying to solve very difficult challenges on your own can only get you so far. Sooner or later you realize the need of help from others. Attending international meetings is a great way to broaden ones horizons and it also proposes the opportunity to collaborate with people from many different countries, who might have a completely contrasting perception of the problem at hand. Hosting such an event introduces a crucial issue to the organizers, because they need to be able to choose the perfect spot for the meeting to maximize productivity without giving the attendants recovery time from their trips for financial reasons.

In our paper we deal with this task by breaking it down into several phases: Study phase, Design phase, Implementation phase. These 3 pillars provide a comprehensive study and solution to the given task. This approach also means well distributed work amongst the team members which implies that everyone has a good understanding of their role.

In the Study phase our aim was to acquire as much information about the topic as possible. Reaching out to various researches a databases we investigated what are the things that really matter when attending a meeting. We researched various factors that might have an impact on the productivity of the meeting, such as jet lag, noise pollution, precipitation etc. That was followed by a series of hypotheses based on external sources. In the final stage of this phase we designed parameterized mathematical functions that describe the exact influence of the considered factors on productivity.

The next step was the Design phase in which we formed a number of algorithms to find a place for the meeting using the relations set in the Study phase. Overall, we considered three methods. The Differentiation method expresses the productivity as a function of the coordinates of a given place and then uses differentiation to find the maximum value. The Zones algorithm divides the Earth into areas based on similar influence of the factors and finds the best meeting point that way. At last we opted for the most accurate Discrete algorithm, which calculates the productivity for more than 7000 cities and ranks them according to this number.

Later, we introduced a secondary, financial criterion, which determines the overall cost of a meeting based on the Cost of Living Plus Rent Index. In this case, the Discrete algorithm was used to rank the cities by cost. In the end, the overall productivity and overall cost were compared to find the most productive and cost-efficient environment.

Finally, in order to successfully test our model on multiple data sets, we created a fully functional program utilizing Java, Git, Maven, and a MySQL database. The code is publicly available at the address in our paper and is also to be found in the appendix. This is what we call the Implementation phase.

Contents

2 Introduction

You and your colleagues are working on perhaps the most salient project you have ever been involved with. The clock is ticking fast and your entire team has to make immensely critical decisions. This is when winners are separated from losers and legends are born. There are a plethora of reasons why things might go wrong and the goal of our paper is to ensure that the whereabouts of your meeting are not one of them.

Organizing events can bring headaches even to experienced coordinators. Given a set of attendants with diverse preferences, ages, and origins, they have to find a place where everyone will do their best. Our paper designs a model which returns an ordered list of locations based on a set of attendants considering a sundry range of factors. The factors are sub-models to the main algorithm. It also cuts the price of the meeting as much as possible while leaving the extent to which productivity will be compromised to the organizers.

The final part of this paper is dedicated to showcasing the architecture and implementation of our algorithm in Java and the integration of a user-friendly UI.

3 Assumptions

Assumption 1. *The individual importance of all attendants is the same.*

We treat all attendants in the same way. Theoretically, some of them could be more important that the other in a way that they are irreplaceable experts in their field. But according to the given problem task, each attendant contributes approximately equally to the end result of their intellectual teamwork.

Assumption 2. *Attendants arrive no sooner than the day before the first day of the conference and have at most one night to rest and start acclimatizing.*

According to the task, the meeting is substantially important and for the simplicity of the problem, we assume that in a real situation no one would be willing to start the meeting not having rested and acclimatized to the environment for at least one night. On the other side, the budget of the organizing company is limited and hence attendants are not able to arrive in advance to rest more.

Assumption 3. *Every place that is suitable for the meeting must be of a certain size in order to comprise a certain level of base infrastructure necessary for the meeting.*

See section 8.3 for more details.

4 Definition of important terms

Attendant	Someone who attends the meeting
Productivity P	The ratio of the amount of work carried out under certain conditions and the amount of work one would meet in ideal conditions. We use logarithmic transformation to express productivity in percentage terms. It follows that in these terms of percentages we use geometric relations, e.g. geometric mean instead of arithmetic average.
Day index D	Index of the particular day of the meeting. $D = 0$ is the day of the arrival (in the time zone of the destination) and $D = 1, 2, 3$ are the first, second and third days of the conference, respectively.

5 Analysis of the problem

The first step in solving this problem will be to analyze it and divide it into subproblems. First of all, we will need a way to compare the viability of two locations. For this reason, it is necessary for us to be able to estimate the productivity of our team as precisely as possible.

Every individual may be affected by the location differently. This may be due to different travel times required to reach the destination, a different number of time zones crossed, or just personal preference. As a result, in order to calculate the performance of the entire team, we first need to determine the productivity of an individual in a given location.

In such a complex situation, there are bound to be several factors influencing a person's productivity. Considering the effect of all these factors combined would be nearly impossible. For this reason, we decided to consider every factor's effect by itself with the assumption, that except for a few simple cases, these factors don't

influence each other. Is it also important to investigate, how the effects of these factors may change throughout the meeting. The total productivity of a person should therefore be the sum of their productivities across the three days.

An important condition in this problem is, that the participants are expected to contribute approximately equally, therefore no single one is more important than another. From this condition, we can conclude, that nobody's work is fully dependent on the work of someone else. In practice, this means that we are going to reach the total productivity by summing up the productivities of all individuals.

Another part of this problem, is going to be designing an algorithm for using the processes described in the previous parts of the paper to find the best locations. In this part, we will also consider price as a secondary priority.

The problem is therefore divided into several subproblems:

- Determine the significant factors influencing an individual's productivity.
- Devise a model for determining the influence of each factor on an individual.
- Combine the models to estimate an individual's productivity in a given place.
- Determine the best locations.
- Offer a solution with the consideration of price.

6 Factors

Moving someone to an international meeting we have to consider a number of factors that might have a subsequent impact on their productivity. Several factors are fairly apparent due to the severity and scope of their influence. Some of these more evident elements are jet lag, temperature difference, or altitude level. Nonetheless, further examination of more subtle aspects is crucial in order to maximize the credibility of our model. Factors such as noise pollution, crime, or rainfall all belong to this group of delicate aspects.

In this section, we will firstly examine which factors to consider in our model. Secondly, we will design a model for each of them. And lastly, we are going to estimate how the effect of an observed sub-model changes over the course of 3 days.

Our sub-models heavily rely on data and research. These sources are sufficient enough to evaluate most relations and properties of the desired models, however, in some cases, there is a lack of adequate information for us to precisely design the effect on one's productivity. For instance, one resource [6] suggests that working under natural light creates an increase in productivity of about 21%. This means that we should consider a function of decreasing productivity for the amount of time spent before sunrise, or after sunset. Yet we can only roughly estimate the scale to which this decrease in lighting will actually take a toll on productivity. That is the reason for our inclusion of parameters in most of the models.

The primary function of these parameters is to create models which are easily customizable. Customization means than any consequent research which goes deeper into the dependency of our factors and productivity can be easily included in our sub-models. Furthermore, using these parameters we can involve attendant input based on their preferences. Information such as their age, country of origin, and rates to which they believe certain factors affect them can be easily obtained by simple psychological surveys. This way we will be able to tailor each model to predilections of our attendants. The parameters we use in the algorithm are based on our research of each influencing aspect.

Each sub-model is a parametrized function returning the adjusted productivity based on a particular criterion.

Parameters	Arguments
attendant	destination
meeting date	day index

Table 1: parameters and arguments of a model function

Here, by the attendant parameter, we mean every information that we decide to investigate (attendant's origin, age, etc.).

Our methodology when it comes to estimating these parameters is based on our own experiences, research, and testing the model.

6.1 Jet Lag

The first and potentially the most severe factor we need to consider is jet lag. It refers to a physiological condition caused by a misalignment of local time with the body's circadian rhythm, a process responsible for

the control of biological processes based on the time of day. As a result, patients most often experience insomnia and daytime sleepiness but the symptoms may also include dysphoric mood and gastrointestinal disturbances [12], all of which cause a decline in productivity.

Research [3] has shown, that this effect only occurs when conducting trans-meridian travel and is dependent on the time difference between the origin and the destination. Traveling within a single time zone, therefore, doesn't induce jet lag. It has also been shown that the symptoms of jet lag are stronger after eastward time zone transition than after westward time zone transition [9]. The severity of these symptoms and the length of recovery needed seem to some extent chaotic, however they follow certain trends, which we based our model on.

The recovery length has been estimated by the Finnish Institute of Occupational Health to be roughly d/L_E days for eastward travel and d/L_W days for westward travel, where $L_E = 2, L_W = 2.5$ and d is the time difference between the origin and the destination in hours. It means that L_E is the number of hour difference one is able to adapt to during one day after travel.

Let P_W denote the theoretical productivity in the first day after traveling east to a place with 12 hour time zone offset providing that you have to recover by shortening your circadian rhythm for the duration of recovery. Then let P_E denote the same after traveling west to a 12 hour time zone offset place providing that you have to recover by temporarily prolonging your circadian rhythm. Note that P_W and P_E are only theoretical productivity values. Even though the destinations when traveling east and west are the same (in a time zone exactly opposite the current location), the difference between P_W and P_E is in the way it expects one's body to recover, by temporarily shortening or prolonging the circadian rhythm, respectively, to settle the time offset.

As the recovery length is inversely proportional to the amount of productivity, we can see that the following equation holds:

$$\frac{P_W}{P_E} = \frac{L_E}{L_W} = \frac{2}{2.5}$$

Suppose we have P_M as a parameter, which stands for the geometric mean of P_W and P_E. Geometric mean is used because we are speaking in terms of percentages.

$$P_M = \sqrt{P_W P_E}$$

From these two equations we can calculate the two values of productivity as follows:

$$P_W = P_M \sqrt{\frac{L_E}{L_W}} \qquad\qquad P_E = P_M \sqrt{\frac{L_W}{L_E}}$$

We calculate the time difference d as the difference between the offsets of the two time zones (origin and destination) from UTC at the specified date of the meeting and then d is equal to the difference between the two offsets. Determining a time zone offset also requires the meeting date, as it also takes into account daylight saving.

Finally, we calculate the loss of productivity that would be caused by letting your body adapt to the time zone change by either shortening and prolonging the circadian rhythm and pick the option with more productivity. Here we assume that the body would choose the more natural way, i.e. with the smallest negative impact on productivity. At last, the equation is combined with D, the day index. By the i-th day ($D = i$), attendant is negatively affected by jet lag as if the time zone difference was decreased by $i \cdot L_E$ or $i \cdot L_W$ accordingly, because he has been adapting to the time zone change for i days with the rate of L_E or L_W.

$$d = \Big|\ \text{(offset of destination from UTC)} - \text{(offset of origin from UTC)}\ \Big|$$

$$L = \begin{cases} L_E, & \text{if traveling east} \\ L_W & \text{otherwise} \end{cases}$$

$$P_O = \begin{cases} P_E, & \text{if traveling east} \\ P_W & \text{otherwise} \end{cases}$$

For simplicity of the final equation, let R be equal to the ratio of the amount of time zone difference in hours that the attendant is still to be adapted to and the constant 12, which stands for the maximum time zone difference.

$$R = \frac{max(0, d - L \cdot (D-1))}{12}$$

Based on the observation and personal experience of the authors, we estimate that the dependency of productivity loss on R is quadratic. The productivity can be then projected with this formula:

$$P = 1 - R^2 \cdot (1 - P_O) = \left(\frac{max(0, d - L \cdot (D-1))}{12}\right)^2 \cdot (1 - P_O)$$

6.2 Distance

There is no doubt you remember having experienced a long exhausting journey after which your only desire was to rest. It describes just the way traveling works - you definitely have to spend more energy on it than on relaxing in your dream destination.

Hence the loss of productivity of the attendants can be said to be proportional to the loss of vigor from the journey which can be gauged as being proportional to the time spent traveling.

$$\text{(productivity loss)} \quad \propto \quad \text{(travel time)}$$

One approach from this point could be to **use** some **web API** for determining possible routes between the given origin and destination, for instance API provided by Google or Bing. This could lead to precise results as the APIs may be able to plan routes with possible flights and bus transfers supported by real time schedules.

On the other hand, this concept would require a huge number of requests towards the web service, considering the fact that we have to obtain routes data one by one for every pair of locations we want to consider. Therefore this kind of a brute-force solution is not suitable without further improvement by dramatically reducing the number of required web requests.

Another method is to estimate the travel time to be proportional to the **distance between origin and destination**. Although this idea can lead to wrong results for some locations, it is fairly useful data resource considering the simplicity of the principle. There are several methods used for determining the great-circle distance between two points on a sphere given their longitudes and latitudes. See Table 2 for their comparison.

	Law of Cosines	**Haversine formula**	**Vincenty's formulae**
Accuracy	lower for small distances	sufficient	high
Nearly antipodal points	well handled	well handled	inaccurate results
Calculation time	normal	normal	slow

Table 2: Comparison of great-circle distance determining methods

Based on the pros and cons of each method (see Table 2), we decided that *haversine formula* best suits our needs.

Furthermore, it is generally known that because of agility and vigor of young people they are less affected by the exhaustion from a journey and even may be more willing to travel long distances in comparison with the elderly. Here we will use parameters P_Y and P_E which tells us the theoretical productivity in the first day after a journey of 20 000 km of the young (age 18) and elderly (age 80), respectively. The distance 20 000 km is chosen according to the fact that the Earth's circumference is approximately 40 000 km and the maximum distance between two points is half of that. Let P_M denote the theoretical productivity in the first day after a 20 000 km journey by a particular attendant. We will use his age A (in years) as follows:

$$P_M = P_Y - \frac{A - 18}{80 - 18} \cdot (P_Y - P_E)$$

Furthermore, since the productivity P is in terms of percentages, we have to map the range $[0, 20\,000\,\text{km}]$ to the range $[P_M, 100\%]$ exponentially and inversely. Let our function $f(x) = P$ for some distance x be of the form $f(x) = e^{-Kd}$ with constant K. So far we know that $f(0) = 100\%$ and $f(20\,000\,\text{km}) = P_M$. With this we can calculate the constant.

$$K = \frac{\ln(P_M)}{-20\,000\,\text{km}}$$

and

$$P = f(x) = e^{-Kd} = e^{d \cdot \frac{\ln(P_M)}{20\,000\,\text{km}}}$$

However, the travel fatigue only affects the first day ($D = 1$) of the meeting. For the following days, the attendants have enough time to catch up with their sleep and mend their energy deficit.

One improvement to this sub-model would be to guess how difficult it will be to travel to a particular destination. We can assess the probability that the city will have an airport, or that the the train or bus station will be accessible with more travel connections. Population is exactly the attribute that helps us estimate the value. The more people live in a city, the more connections should the city afford.

We can not easily determine the probability of a city having an airport, but we can increase the productivity proportionally with the population to increase the chance that the city will be chosen as one of the most suitable destinations. For this we can use the following formula with parameter P_P telling the increase of productivity in percentages per a certain amount of increase in population. p denotes the population of the city. We use square root of p so that the population difference between extremely large cities doesn't make so huge difference in productivity as opposed to a difference between smaller cities.

$$P = f(x) + P_P \cdot \frac{\sqrt{p}}{10^6}$$

where

This introduces an exceptional factor in the way that it can return productivity higher than 100%. This is because we cannot identify one's ideal productivity when evaluating population of cities, as there is no maximal restriction of population.

6.3 Altitude

The brain of human relies on oxygen which implies that the level of oxygen in air may influence mental performance. The density of oxygen is proportional to the density of air, which depends on pressure and temperature. After evaluating temperature and pressure as functions of altitude, we can conclude a formula describing air density.

The temperature at altitude h meters above sea level can be approximated by the following formula (valid inside the troposphere):

$$T = T_0 - Lh$$

where T_0 is the sea level standard temperature and L is the temperature lapse rate.

The pressure at altitude h is given by:

$$p = p_0 \left(1 - \frac{Lh}{T_0}\right)^{\frac{gM}{RL}}$$

where p_0 is the sea level standard atmospheric pressure, g is the earth-surface gravitational acceleration, M is the molar mass of dry air and R is ideal gas constant.

According to the *International Standard Atmosphere* the standard values are as follows:

$$\begin{aligned} T_0 &= 288.15\,\mathrm{K} \\ L &= 0.0065\,\mathrm{K/m} \\ p_0 &= 101.325\,\mathrm{kPa} \\ g &= 9.806\,65\,\mathrm{m/s^2} \\ M &= 0.028\,964\,4\,\mathrm{kg/mol} \\ R &= 8.314\,47\,\mathrm{J/(mol \cdot K)} \end{aligned}$$

Density can then be calculated using a molar form of the ideal gas law:

$$\rho = \frac{pM}{RT} = \frac{Mp_0\left(1 - \frac{Lh}{T_0}\right)^{\frac{gM}{RL}}}{R(T_0 - Lh)}$$

Studies [13, 4, 5] confirm that increased amount of oxygen improves the cognitive performance of your mind. According to [4], heart rate decreases with more concentrated oxygen in the air, as the blood cells carry more oxygen molecules. Following that, we can assume that lower air density, well known for causing a decrease in heart rate, causes the opposite effect and lowers your mental performance.

Now we can use the results from various performance tests found in [11] to determine how cognitive functions of a human are influenced by lower air pressure. The function that best fit the points of data from this research ended up being a logarithmic curve of the function:

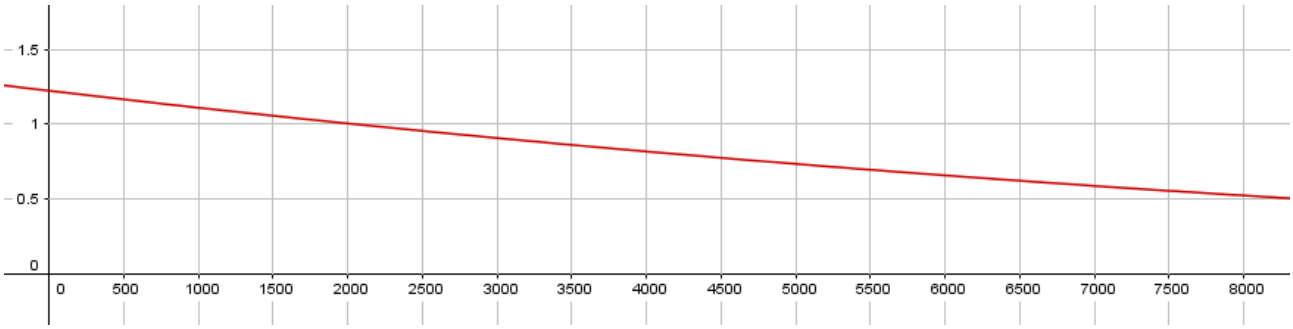

Figure 1: Air pressure based on altitude

$$f(x) = log_a\,(x - b) + c$$

$$a = 3.995$$
$$b = 0.7576$$
$$c = 14.05537$$

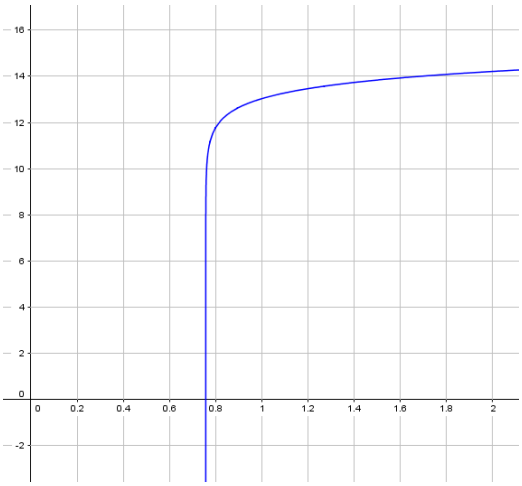

Figure 2: Human productivity based on air pressure

Finally we need to combine both aforementioned functions:

$$g(h) = f(\rho) = log_a\,(\rho - b) + c = log_a \left(\frac{Mp_0 \left(1 - \frac{Lh}{T_0}\right)^{\frac{gM}{RL}}}{R(T_0 - Lh)} - b \right) + c$$

Using the variables mentioned earlier we can estimate a person's productivity in a certain altitude by dividing the value of $g(h)$ by the value of $g(0)$, in other words, we will take a person's productivity at sea level as the baseline.

$$g(0) = log_a \left(\frac{Mp_0}{RT_0} - b \right) + c$$

$$P = \frac{g(h)}{g(0)} = \frac{log_a \left(\frac{Mp_0 \left(1 - \frac{Lh}{T_0}\right)^{\frac{gM}{RL}}}{R(T_0 - Lh)} - b \right) + c}{log_a \left(\frac{Mp_0}{RT_0} - b \right) + c}$$

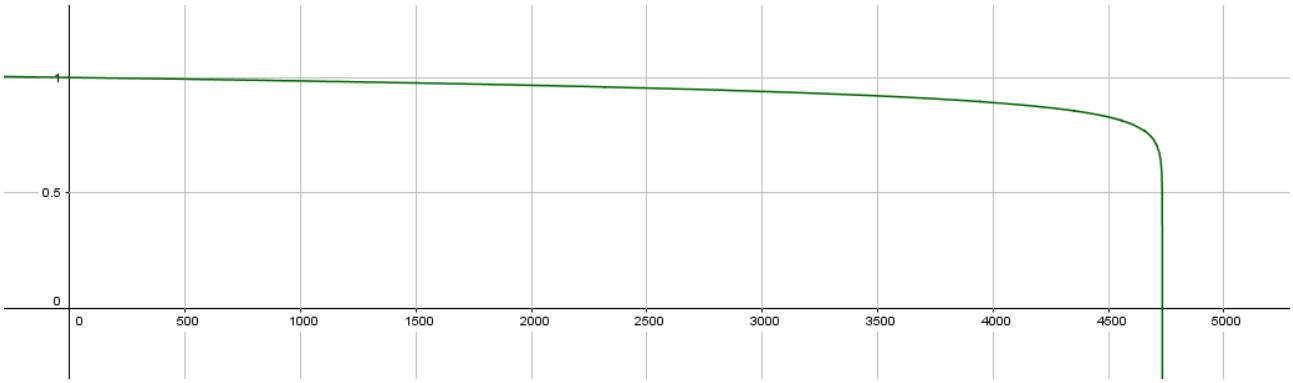

Figure 3: Human productivity based on altitude

A disadvantage of this approach is that because we used a logarithmic function to estimate the effect of air density on cognitive function our model behaves erratically for altitudes above 4600 meters above sea level. In spite of that, we can safely use this model, because altitudes above 4600 meters can be considered unsafe for an untrained person with no option of slow acclimatization and so such cases do not have to be examined.

When evaluating this factor, we will consider the effect on human performance on different days of the meeting to be equal. This position can be justified by the fact, that it may take considerably more time for a person with no previous experience of altitude acclimatization to get used to the new environment and even then the effects on a person's mental ability are likely to persist in some form. Our model may be further improved in this area by considering information from our attendees, about their previous high altitude experiences, however, currently not enough research has been done about these effects to create a more precise model.

6.4 Daylight

A number of studies suggest, that lower sunlight levels lead to lower productivity. That is mainly caused by the release of the natural hormone, Melatonin, which is directly affected by the amount of sunlight one receives.

A professional consulting company, the Heschong Mahone Group, conducted a study on grades and sunlight exposure in the elementary school classrooms. According to their studies [6], among twelve models considered, there is a central tendency of a 21% improvement in student learning rates from those in classrooms with the least amount of daylight compared to those with the most. From this data, we can say that a person working in bad lighting conditions, such as purely artificial lighting experience a drop to $100\%/121\% \doteq 83.\overline{3}\%$ of their productivity compared to ideal conditions. In our model, the difference between productivity in daylight and nighttime is based on this data.

An average workday consists of usually 8 working hours (our model allows the user to set a number of working hours). In order to correctly determine the influence of daylight, we need to calculate the amount of time the sun is up on an average workday and then apply the data from the study of the Heschong Mahone Group.

Naturally, this model does not take into consideration that in a particular place a day may, in fact, end up being shorter because of obstructions, such as hills and mountains. However, taking these differences into account for every area that is being evaluated is virtually impossible. Our model can at least give us a useful approximation of the real length of day and it's effect on a person's cognitive functions.

To calculate the times of sunrise and sunset we will use the sunrise equation:

$$cos(h) = -tan(\phi)tan(\delta)$$

Where ϕ is the geographical latitude of the place in question, δ is the sun declination, a value that changes throughout the year and can be calculated from date and axial tilt, for this we will use a value from a precomputed table based on the day of the year. Here h is the hour angle, which can be calculated as $30° \times (T - 12)$, where T is the local time in hours. It is also worth nothing, that angles in geographical variables are usually expressed in degrees, rather than radians. Therefore, we will consider all angles in this section to be expressed in degrees.

$$h = \pm arccos(-tan(\phi)tan(\delta))$$

This equation gives us two possible values for h with the positive one being the hour angle of sunset and the negative being the hour angle of sunrise. We can then convert these angles into time as follows:

$$\text{Time of sunrise: } 24 \cdot \frac{-arccos(-tan(\phi)tan(\delta))}{360°} + 12$$

$$\text{Time of sunset: } 24 \cdot \frac{arccos(-tan(\phi)tan(\delta))}{360°} + 12$$

The next step will be to calculate the ratio R_L of how much time designated as working hours is spent before sunrise or after sunset and the full length of working hours.

$$R_L = \frac{\text{working time during nighttime}}{\text{total working time}}$$

Given the parameters of start and end times of work, this calculation becomes quite trivial. The final productivity of a person in such an environment will be:

$$P = R_L \cdot 0.8\overline{3} + (1 - R_L) \cdot 1$$

6.5 Temperature

In order to choose the best working conditions for a group of people, it is necessary to establish, whether the climate of a particular place plays any part in determining an individual's productivity.

The first aspect of a climate that we are going to be examining is its temperature. A number of studies [15, 10] have shown, that temperature does, in fact, have a considerable impact on human cognitive functions. One scientific paper in particular, however, has shown to be an invaluable source, when it comes to determining the extent of this impact. A NBER working paper from 2013 [2] has studied the correlation, between productivity and indoor temperature (Figure 4). This research presents us with several important observations. Firstly, the ideal temperature for mental work is around $22°C$. We can also use the data measured in this study in order to estimate the function to determine the productivity of a person working at a given temperature.

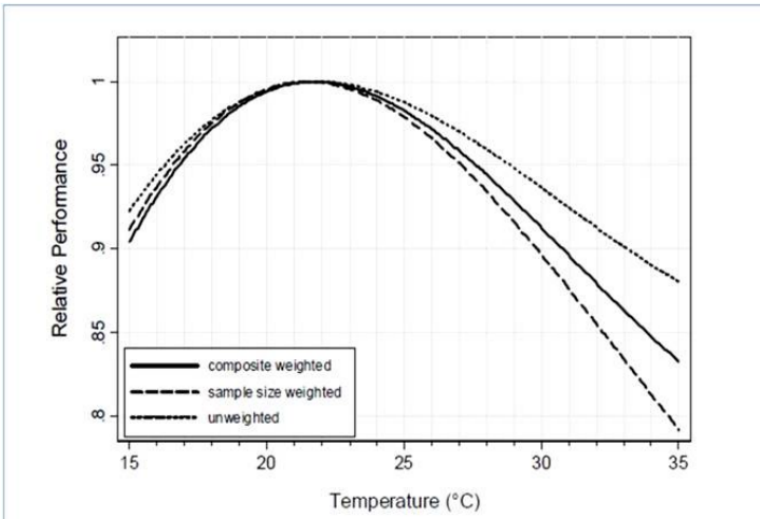

Figure 4: Task performance vs. temperature [2]

Using the least-squares fit we arrived at an approximation of (Figure 5):

$$P \approx -1.06907 \cdot 10^{-7} \cdot t^4 + 0.00003 \cdot t^3 - 0.00344 \cdot t^2 + 0.11109 \cdot t - 0.08269$$

where t is the temperature of the destination.

It is pretty effortless to obtain some data upon which temperature impact can be estimated. We used historical temperature data per month derived from the Climate Research Unit (Mitchell et al, 2003), aggregated by countries.

This approach does, however, have its shortcomings. The first problem we have to consider is that the original research only studied the effects of indoor temperature. In our case, the indoor temperature is not so dependent on a particular city, since it can be easily controlled by heating or air conditioning. We can, however, predict, that outside temperature is going to have an effect on productivity, whether it be by decreasing comfort while traveling, or in case the outside temperature cannot be entirely differentiated from the indoor one, because

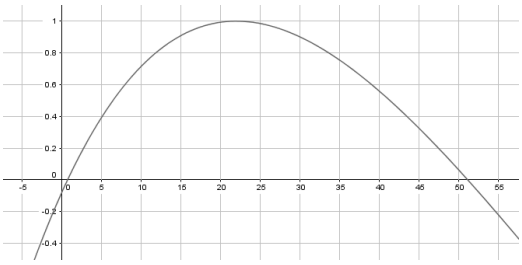

Figure 5: Task performance vs. temperature, function fit

of insufficient heating, insulation etc. We therefore decided to model the relationship between productivity and outside temperature based on the model for indoor temperature, since we can safely assume that both models behave similarly. To estimate an individual's productivity, in this case, we will use the function mentioned above, using a parameter, that will determine the magnitude of performance impact.

$$P = \big((-1.06907 \cdot 10^{-7} \cdot t^4 + 0.00003 \cdot t^3 - 0.00344 \cdot t^2 + 0.11109 \cdot t - 0.08269) - 1\big) \cdot K_T + 1$$

where K_T is the temperature effect parameter. For the purposes of this model, we estimate this value to be around:

$$K_T \approx 0.05$$

It is also worth noting, that according to our research, no considerable adaptation to local temperature can occur in such a short period of time. Our model considers the effect of temperature to stay the same throughout the meeting.

This simple approach, however, may not be precise for large countries where the temperatures significantly differ for different locations within the country. Further improvement of the approach could be made by obtaining more precise data not aggregated by countries.

Additionally, the month of the day with appropriate day index D is used to determine the average temperature in the country for the given day of the meeting. An enhancement could base the temperature prediction not only on the month but also on the day of a month and the average temperatures in the previous and following months.

6.6 Air humidity

Another aspect of a local climate that can heavily influence the ability of an individual to work in a particular environment is the relative humidity of the surrounding air. Humidity directly influences the heat index, a metric that determines the human-perceived temperature, given objective air temperature and relative air humidity. In order to implement humidity into our model, we only need to calculate the heat index for a given location in a particular time and use it as an argument for our temperature model instead of objective temperature. Heat index HI can be approximated using several formulas. We decided to use the most widely used one. This formula uses the Fahrenheit scale, so we also need to convert the dry-bulb temperature into Fahrenheit and the resulting value back to Celsius.

$$HI_{\circ F} = c_1 + c_2 t_{\circ F} + c_3 R + c_4 t_{\circ F} R + c_5 t_{\circ F}^2 + c_6 R^2 + c_7 t_{\circ F}^2 R + c_8 t_{\circ F} R^2 + c_9 t_{\circ F}^2 R^2$$

$HI_{\circ F}$ – heat index in degrees Fahrenheit

$t_{\circ F}$ – dry-bulb temperature in degrees Fahrenheit

R – relative humidity expressed in percents

$c_1 = -42.379$	$c_4 = -0.22475541$	$c_7 = 1.22874 \cdot 10^{-3}$
$c_2 = 2.04901523$	$c_5 = -6.83783 \cdot 10^{-3}$	$c_8 = 8.5280 \cdot 10^{-4}$
$c_3 = 10.14333127$	$c_6 = -5.481717 \cdot 10^{-2}$	$c_9 = -1.99 \cdot 10^{-6}$

Now we can use the resulting value of $HI_{\circ C}$, heat index in degrees Celsius, as an argument for our temperature model.

Unfortunately, we haven't managed to find sufficient data with average humidity values. Hence the temperature model only uses the real temperature values instead of calculated heat indexes.

6.7 Noise and light pollution

There are a few distinct ways to interpret what noise is. Noise can mean unwanted fluctuations in electrical signal due to seemingly random movement of electrons. This definition is used in audio processing where different algorithms are applied to smoothen this undesired event. Usually, there do exist software solutions that filter out these undesirable effects, but for certain noises, we can not rely on software, we have to entrust our brain.

An Oxford University study [14] points out how noise pollution plays a negative role in various aspects of our lives. There is a number of psychological, mental and cardiovascular symptoms which occur mainly due to longer disruptive sound exposure. For our model however, our main concern should be the cognitive effect which is directly linked with annoyance and disruption. The more disrupted someone is, the more their ability to execute a task reliably decreases and ergo they become less productive.

Another type of rebarbative *noise* can be found in the form of light. Light pollution is believed to be the main cause for a considerable drop in Melatonin, which also means worse and shifted sleep patterns and even is connected with cancer, namely breast cancer [1]. Particularly the effect on sleep might noticeably shape one's productivity.

According to the Weber-Fechner law, there is a logarithmic relation between biological stimulus and the perceived sensation. We can notice this phenomenon when we push a finger against our palm. We are much more sensitive to the change of force in reaction to slight taps than to forceful pushes. This makes perfect sense since our sensitivity is the derivative of this logarithmic function. We could consider some relation between annoyance and brain sensitivity which would lead to the dependency between sound wave energy and annoyance, nevertheless, there is a lack of sufficient data when it comes to the average amount of decibels per city or per country.

Moreover, figuring out the model for light pollution leads to similar problems. Even though there is a great amount of research done on sleep deprivation and Melatonin levels, there is virtually no data telling us about night light in different areas of the world.

There is a company called Numbeo which crowdsources data about countries/cities. One of the factors they include in their API is noise and light pollution. The API provides a number ranging between -2 and 2 which tells us to what extent people are annoyed by the observed factor in their respective countries/cities. Using this input we can model the influence on one's productivity. Based on the definition of productivity and on the fact that the scale we get from the API is linear, the function has to be exponential. The function which maps the API output to productivity is the following function of x, defined on $x \in \langle -2, 2 \rangle$:

$$P = f(x) = P_{min}^{\frac{2-x}{4}}$$

where P_{min} is a parameter which tells the productivity in the worst case scenario (when $x = -2$) thus $P_{min} \in \langle -0, 1 \rangle$. Furthermore, our function always fulfills $f(2) = 1$ since for any place where the noise and light pollution causes no harm we should consider a maximal productivity of $P = 1$.

There are no evident sources that prove how the effect of these factors on individuals changes over time. Of course, in the long run, they can mean severe health risks, however, in a scope of three days this change is practically undetectable.

The shortcomings of this approach are that people from different countries with diverse mentalities might react differently to light and noise pollution and that for some countries there is a relatively small number of respondents, however the status quo of reliable data on actual noise and light around the world in nonexistent or lacking. Once we find a better source of data, we can easily include the information, which will improve the precision of our model.

6.8 Quality of life

The average quality of life varies from country to country. Whilst some countries flourish in fairly constant prosperity, other countries experience a humanitarian crisis. The quality of life index designed by The Economist Intelligence Unit combines subjective and objective indicators throughout different factors to create an index which tells the overall quality of life country by country. Although The Economist Intelligence Unit reaches out to a large number of countries, some of their indicators do not have a substantial influence on productivity. Examples of these are community life, family life, and job security.

In the look for a more relevant index, we have again come across Numbeo. The Numbeo quality of life index takes into account factors such as crime rates, safety, traffic, air pollution and water pollution which we believe are more relevant when it comes to productivity than the factors considered by The Economist Intelligence Unit.

The sole purpose of this factor in our model is to provide a country and city ranking which would somehow tell us more about the comfort and peace our attendants would be surrounded by. Due to the lack of data provided by Numbeo, we will consider both countries and cities. Where there will not be data for a city, we will consider the country's index and where we will not find data for a country, we will find the most probable range for our desired index by looking at the given continent's country index values.

Given a positive integer n and a set of life indexes L in a continent, where $n < |L|$, we construct $\lceil \frac{max(L)}{n} \rceil$ intervals starting at 0 with a step of n and for every value in L put it in the appropriate interval. The most probable interval will then be the one which encloses the highest number of index values. Now that we know the most probable range of index values we can estimate the desired index to be the average of these values. It is important to note that with a higher value of n also the precision of our estimation increases, however, we have to cautiously pick n so that the most probable interval will exist. We find this approach more accurate than estimating one value for the whole world as index fluctuations are much less significant in each continent. Once we will have more data, we can easily add a country/city to our sub-model.

The next step is to design a function which maps the linear scale of indexes to productivity percentages. We can take a similar approach as with noise and light pollution. Our function, defined on $x \in \langle min(L), max(L) \rangle$, where we already know that $min(L) = 0$ (since for instance Venezuela has an index of 0 and there cannot be a negative index) will be:

$$P = f(x) = P_{min}^{\frac{max(L)-x}{max(L)}}$$

where P_{min} is a parameter which tells the productivity in the worst case scenario (when $x = 0$) thus $P_{min} \in \langle -0, 1 \rangle$. Notice that $f(max(L)) = 1$ since for the best index we need to consider maximal productivity.

It is not common for cities/countries to make sudden changes which would influence the way we live hence we can say that the actual index over the course of three meeting days is constant. In our model, we also regard the acclimatization to this factor nonexistent since we consider the length of the meeting too short for any applicant to get used to a new quality of life.

The model we have come up with is similar to the one we use for night and light pollution thus very similar shortcomings apply. The crowdsourcing technique used by Numbeo in some cases means quite interesting results, however it is lacking in the subjective aspect. Ideally, a similar research as The Economist Intelligence Unit has conducted with factors more relevant to productivity would fit this model the best.

6.9 Diseases

When traveling to a foreign country, we need to consider the differences between our destination and our home country from the health perspective as well. These differences are present in many different areas. For instance, drastic change in cuisine can cause severe digestion problems or certain areas may be more violent etc.

Being in a perfect shape is very important in order to sustain a high level of productivity. That is why our model should also take this factor into account. The ideal way to do this is to identify diseases which have an incubation period of 3 days or less (so that the disease influences the productivity) and calculate the probability of someone getting infected.

The most reliable source for International comparisons is the World Health Organization because it is standardized for cross- cultural comparisons. It is also reviewed annually, which often makes it more current than individual country data that can take years to compile.

The problem with incorporating diseases into our model is the calculation of the probability of someone getting infected. Even though we have some data available, for many diseases it is incomplete, especially in third world countries and that could cause serious issues when determining the best place to hold the meeting. In addition to this, we are well aware of the fact that only a small number of diseases have an incubation period of 3 days or less, so this factor would not have a huge impact on the productivity either way. That is why we have decided to exclude it from our model.

6.10 Precipitation

Bad rainy weather, in general, induces a negative mood, so naturally, we feel that our work productivity should be decreased by its effect. This fact is supported by a recent study [8], which shows that about 82% of the respondents thought that weather conditions would increase productivity, and about 83% responded that bad weather conditions would decrease productivity.

However, the same study suggests that on a sunny day people are tempted to think about doing outdoor activities and they tend to be less focused on their work. This hypothesis was tested by assessing worker productivity using archival data from a Japanese bank's home-loan mortgage-processing line. The data includes

information on the line from the roll-out date, June 1, 2007, until December 30, 2009, a 2.5-year time period. In total 598,393 transactions made by 111 workers were examined.

According to the results of this study, a one-inch (25.4mm) increase in rain is related to a 1.363% decrease in time required for each transaction, which leads to better productivity. Also, the error rates were very small (less than 3%) and the variation was little, so we can consider the decreased time the only change in productivity.

We can use this data to estimate the effect precipitation has on a worker's productivity. Firstly, we will express the relation between the increase in rain and the reduction of time needed to complete a task. The data in the aforementioned research suggests that if $t(x)$ is the time needed to complete a task in x mm of rainfall, then:

$$t(x) = (100\% - 1.363\%) \cdot t(x - 25.4)$$

From this, we can determine that this relation can be estimated by an exponential function:

$$t(x) = (100\% - 1.363\%)^{\frac{x}{25.4}}$$

Now we have to estimate the productivity based on time needed to finish a task with a given precipitation. The more productive a person is, the less time they need to finish a task. We can therefore say, that a person's productivity is inversely proportional to the time.

$$P = \frac{1}{t(x)}$$

$$P = (100\% - 1.363\%)^{\frac{-x}{25.4}}$$

According to the research, we base our theory on, the higher precipitation is, the more productive a person becomes. Our definition of productivity says it's a ratio of the amount of work done under particular conditions and the theoretical amount of work under ideal conditions. But here, the ideal conditions are unknown, or more precisely, they don't exist - for every x_0 mm of rainfall, we can imagine an environment with $x > x_0$ that is, according to the formula, potentially better, resulting in higher productivity. Hence we cannot say that any certain level of productivity is the maximum possible. Our model for this factor will therefore be an exception being able to return a value of productivity higher than 1. Nevertheless, it is not a problem at all, it only violates the definition of productivity that is used in other models.

For the research has been done on a relatively small amount of rain, it only tells us the equation holds for small values of x. However, our data show that in certain countries, the precipitation is extremely high. To counteract the bias caused by evaluating the equation in extreme cases when it rains much more, we will substitute $\sqrt{x}$ for x. This allows us to handle small values of x approximately correctly while nicely coping with extreme scenarios of rain when monsoon type raging storms happen. Further, we introduced a parameter (coefficient) K to change the magnitude of the effect if needed.

$$P = (100\% - 1.363\%)^{\frac{-K\sqrt{x}}{25.4}}$$

Precipitation is a factor that reflects the entire climate of a certain area. It is highly unlikely that someone coming from a different climate would get used to it over the course of three days, so this aspect is not considered in our model.

7 Alternative solutions

The purpose of the previous section was to evaluate the effect of the proposed factors on attendants, yet if we were to forget about the constrains of the problem for a while then we could find alternative solutions which would minimize some of the factors.

When it comes to jet lag there are efficient ways to quickly recover and make the effects almost undetectable. An app called Sleep as Android gets people gradually used to the shifted sleeping patterns days before travel by a revolutionary method called *intermittent light therapy* [7]. The method and why it works is described in more detail in this article about a modern way to solve the jet lag problem [16]. There, we can learn about how jet lag treatment is getting to new dimensions where people will not be affected anymore. Furthermore, the influence of the light pollution factor can be lowered by using sleep masks. Theoretically, people could also wear oxygen masks to eliminate the significance of the altitude aspect.

Finding alternative solutions is not the goal of this work, however we believe that for the purpose of the meeting, attendants should be aware of them.

8 Algorithm

Now that we are able to calculate a single person's productivity in a given location, it is time to combine the results for our attendants and choose the best possible location. We considered several possible algorithms for reaching our final answer, all with their own advantages and disadvantages.

8.1 Differentiation

The first algorithm we examined was a method based on calculus. The thought process behind this approach is to express the productivity of our team in a given place as a function of the coordinates of a given place, with all the other information, like date, place of origin of our attendees and their age. Then we could use differentiation to find the maximum value and so find the result in a constant time.

We know that we can express the team's performance as the sum of the individuals' performance. We can also express an individual's performance if we have information about the place in question (e.g. temperature, time zone ...). However, we soon realized that not every aspect of a place can be expressed through its coordinates.

For example, if we wanted to express the time zone of a place as a function, it would have to be a discontinuous function because of the sudden changes between time zones. We could use a continuous function to only approximate a place's time zone, however, we would end up with huge errors in the border regions. This problem becomes even more pronounced with qualities such as temperature, humidity, or life quality index.

After some research, we decided that these factors had too big of an impact on a person's productivity to omit and even if we did simplify them to a usable level, it would only create huge errors. As a result, we rejected this method.

8.2 Zones

The next alternative we considered was to choose several of our factors, that have the biggest influence on productivity and divide the world into zones, so that a single zone is homogeneous according to these primary factors and that factors of each zone would cover the whole world. We could then search through the zones, in order to find the one with the best conditions according to the primary factors.

This method gives us an approximation of which zone should be the best according to the primary factors. Then we could "zoom in" to the found zone and examine the cities located in it based on secondary factors.

However, we have found some flaws in this approach. First of all, what should be considered as a primary factor? Even if we created two sets of factors labeled more important and less important their significance could vary depending on attendant input. Moreover, assuming the relevance of each factor would not be influenced by attendant input then still there would be more important factors which could not cover the entire world in zones, for instance the daylight factor, or altitude factor.

Furthermore, if we decided to pick the factors with the ability to cover the whole world as the more important ones, then there is still a chance that after applying the second round of factors there would be a place in the world that would be more ideal with respect to both rounds of sub-models. For these reasons we chose to not use this model.

8.3 Discrete algorithm

The zoning algorithm turned out to be too imprecise and the only way to overcome it's limitations and achieve higher precision is to consider all factors for every area examined. Furthermore, we wanted to prevent a situation, where an ideal location would be lost in an unfavorable area, if possible. This led us to the need of examining every potentially suitable place by itself and not as a part of a larger area.

At first glance this may seem impossible, since there is an astronomical amount of places on Earth. Thankfully, we can limit our search somewhat, since a meeting cannot take place just anywhere. A certain level of base infrastructure is necessary to achieve the best working conditions and productivity. As a result, we will limit ourselves to only examining areas with urban development. Furthermore, we will filter out all municipalities without sufficient population. This value can be adjusted for the needs of the exact meeting, but for now the threshold is going to be set at around 50 thousand inhabitants. This limit should ensure good accessibility, accommodation and services. Now we are left with roughly over 7000 cities around the world to choose from.

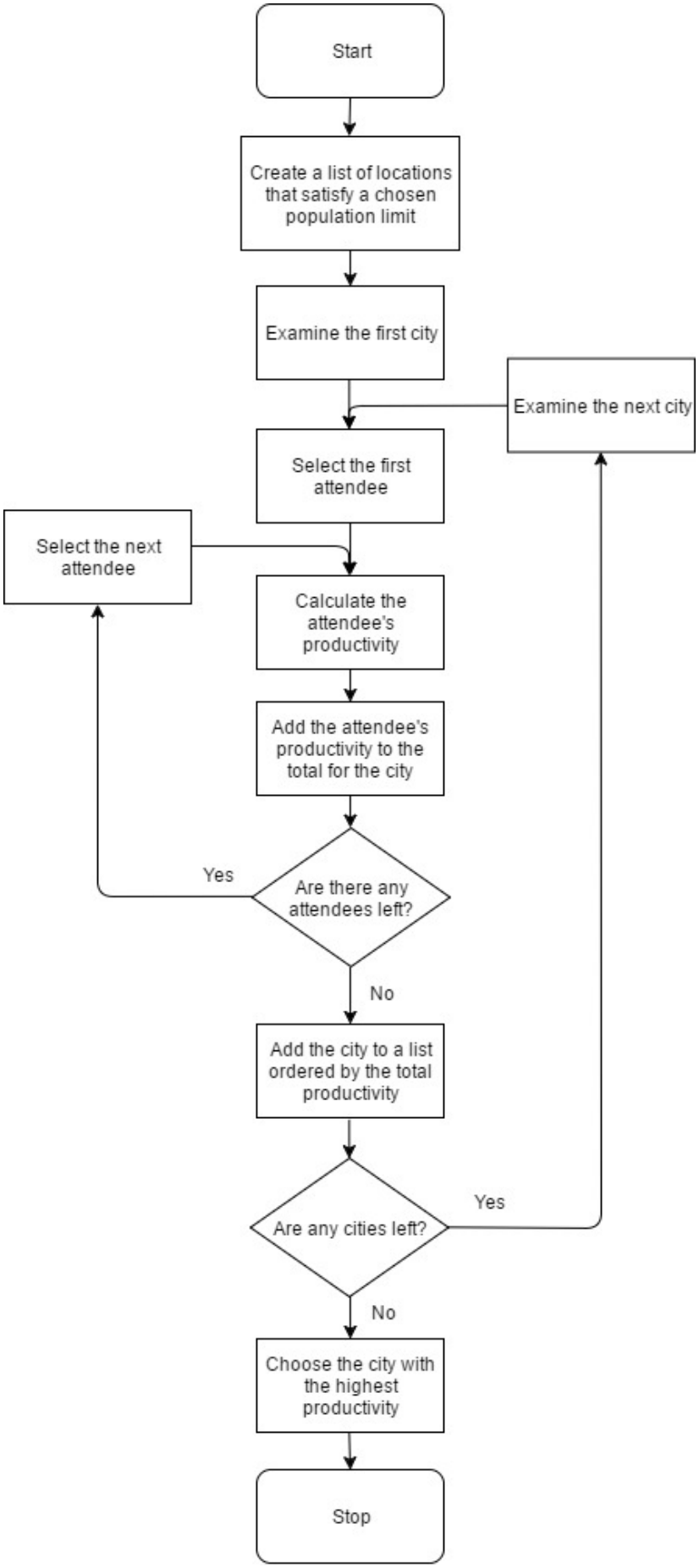

Figure 6: The flowchart representation of our discrete algorithm

We can calculate the productivity of every attendant of the meeting in every examined city using our models for each considered factor separately for each of the three days. Next, we will sum up the productivities for every city and compare them. The result of this algorithm will be a list of cities above our threshold ordered by their respective productivities.

In other words, our algorithm will iterate through all cities, then through all days of the meeting, then through attendees, and finally individual factors. This process may seem very resource intensive, however, thanks to the filtering of possible places we implemented, the computation is almost instantaneous.

9 Secondary criterion

So far, the only criteria we have considered are related to productivity. This method of finding the perfect spot for holding the meeting is great from a mathematical perspective, but in real life, there are other criteria that may influence the decision besides maximum productivity. The most significant non-productivity based criterion is the overall cost.

Determining the overall cost of such an event is rather problematic because there are many variables that can differ from place to place, such as accommodation, food, travel etc..

One way to globally compare cities and countries in this aspect is by using the Cost of Living Plus Rent Index, which is an estimation of consumer goods prices including rent in a certain city comparing to New York City. This means that for New York City, the index should be 100(%). If another city has, for example, Cost of Living Plus Rent Index of 120, it means that the given city is 20% more expensive than New York City. This means, that to calculate the overall cost of living in a city (O), we would need to multiply the Cost of Living Plus Rent Index (C) divided by 100 (because it is given in % relative to NYC) by the overall estimated cost of living in New York City (1,054.81€): $O = \frac{C}{100} \cdot 1054,81$.

Now our aim is to rank the same cities that we used in the productivity-type ranking based on the Cost of Living Plus Rent Index. To achieve this we are going to use the Numbeo database, but due to lack of data for some cities, we are going to consider the Cost of Living Plus Rent Index of the country. To determine the index of those cities, which are not in the Numbeo database nor their country is in the database, we are going to use the same method as explained in section 5.8 - Quality of life index. After allocating an index to every city we calculate the overall cost of living (O) using the formula above.

We need to realize that cost is just a secondary criterion, which to some of us can matter more than to others. While in theory the best place to hold a meeting might be the one with the highest productivity, in practice the organizers may not be willing to pay astronomical sums of money for a few percentile increases in productivity. We could say that everyone values their productivity differently. To find the ideal place to hold the meeting, the organizers need to determine the *cost* (i.e. value in terms of money) of 1% of productivity. By doing that, we can calculate the financial value of productivity for each city, which means that we can compare it with the overall cost (O). For that, we are going to use a parameter (p), which has to be set by the organizers. The productivity cost(C_p) is calculated the following way:

$$C_p = P \cdot p$$

The place to hold the meeting will be the one which has the highest productivity for the lowest price, so in other words, the city which has the largest difference between productivity cost and overall cost ($C_p - O$).

10 Strengths and Weaknesses

We are convinced that our model provides an accurate estimate of where the meeting should take place. It not only considers the most influential factors but also fairly subtle elements such as noise and light pollution. While being precise it likewise comes with a number of perks:

+ It is fast. The algorithm we use is linear to the number of places considered.
+ It is stable. For a set of inputs it always gives the same output.
+ It is easy to customize. Using the parameters we can easily alter the sub-models in accordance with attendant preferences and further research.
+ It is guaranteed that the algorithm finds the best solution possible, as opposed to algorithms which may only find one of the best solutions or their behavior is based on probabilities.
+ Addition of extra sub-models is simple.
+ Certain factors can be easily omitted based on attendant inclinations.
+ The extent to which cost is a secondary criterion depends on the organizing committee.

While we believe in the correctness of our model, there is always room for improvement. We are considering the following points as weaknesses and thus possibilities for future development:

- Our model is restricted to a set number of cities. While we believe we have justified the reasoning behind this decision we consider an algorithm which is able to effectively map any point on the earth with respect to various factors superior.
- We had to estimate data. Due to lack of data, we had to do estimations which decreased the credibility of our model.
- The vast number of parameters makes it harder for organizers to correctly tailor each sub-model.
- There is no implemented way of automatically processing attendant feedback and in this way developing the model.

The future of our model can be broken down into major three parts. Firstly, we need to find more and better data. Secondly, we shall improve the sub-models and make them easier to modify by the organizers. Finally, we should design a uniform and effective way of receiving information from the attendees directly about their preferences and implement an automated way of evaluating it, possibly with the use of machine learning algorithms, to assess attendant feedback.

11 Implementation

We have implemented a fully functional program that works from user putting in input to writing the output. We utilized the following technologies:

- **Java**
  As for the programming language, Java has been chosen for numerous reasons.
  - Java is an object-oriented language which incredibly goes with the modeling principles of this paper.
  - We wanted a high-level language to be able to fully separate our program from the low-level parts so that the models are not dependent on architecture.
  - With Java, we are able to use Maven for effortless dependency management.
  - Robustness and security are benefits implied directly by the language's design. Java puts significant emphasis on early checking for possible errors and so compilers are able to detect many problems that would first show up during execution time in other languages.
  - Much of the code we would write has already been implemented by someone and we can just use it in form of libraries.
  - Multithreading is easily achievable in comparison with other languages and provides the ability to simultaneously perform tasks such as computing, serving GUI, communicating with a database, managing resources, and garbage collecting.
- **Git**
  One of the most important benefits of a version control system is traceability, providing with the ability to explore history. We use Git since it is one of the most common version control systems, offers a great tool-set and vast majority of open source software uses it because of the community using it. Another advantage of employing Git is the possibility to host the code on GitHub.
- **Maven**
  Maven is a software project management and comprehension tool. It can manage a project's build, dependencies, reporting and documentation from a central piece of information contained in one file. Once you introduce Maven into your project, building it and including project libraries can never be simpler anymore.
- **MySQL database**
  We use a database for storing and retrieving various values, e.g. coordinates of the cities, time zone data, average temperatures, sun declination values etc.

Figure 7 is an UML diagram presenting classes of the program that have the most important role.

Our code is publicly available at the following address:
https://github.com/jjurm/meeting-point-planner

The hash of the final commit is `d6b4937bd784a0977ea1b6808f83e7d98bf67576`

You can find the full listing of our code in Appendix A.

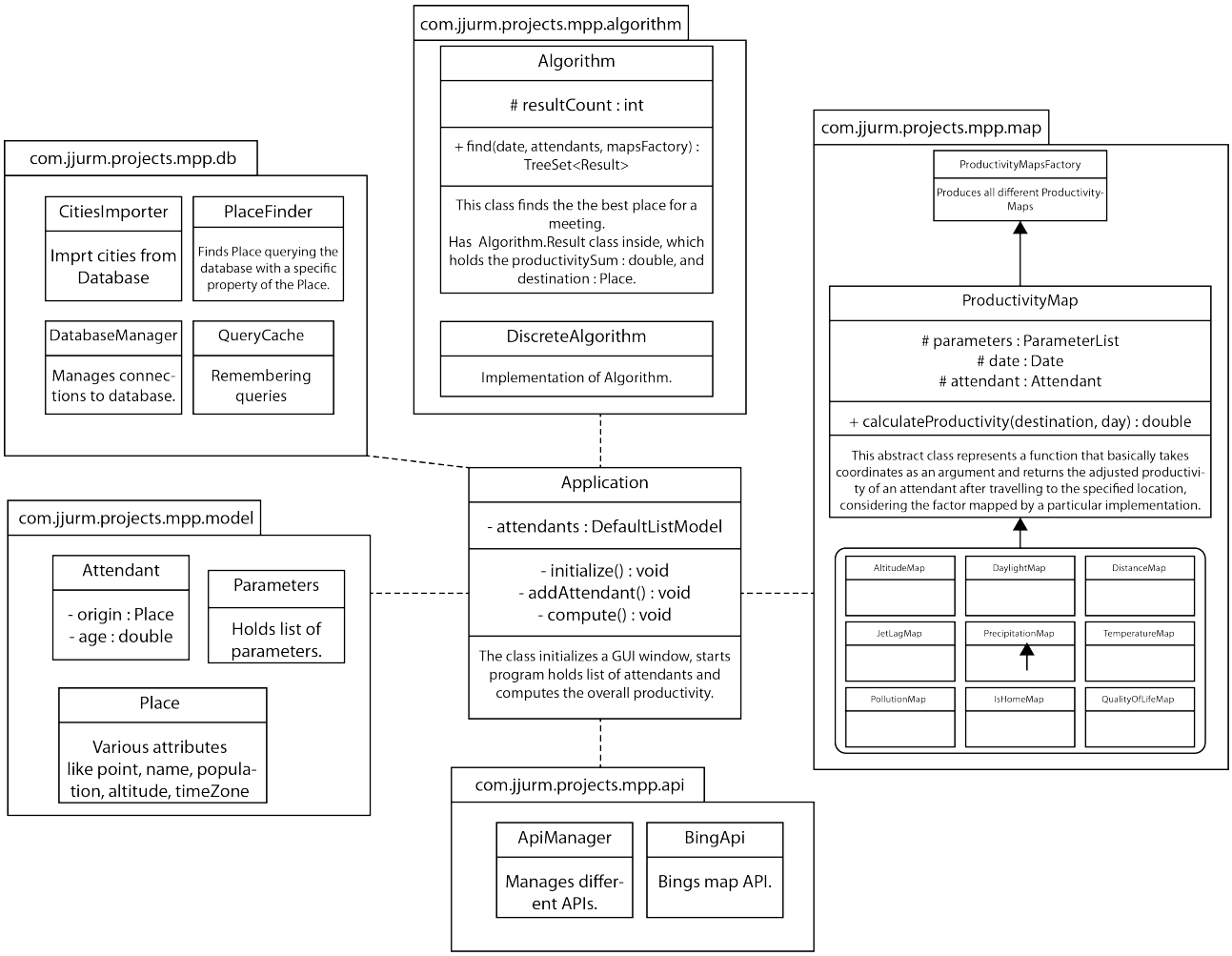

Figure 7: UML diagram of important classes

12 Test cases

Now that we have our algorithm implemented the one thing left to do is to test it on multiple data sets. This process is very important, because it gives an objective image of what our model is capable of. As for the data sets, we are going to be using those provided in the task.

Since we are not given any information about the age of attendants, to prevent biased results we assumed their age to be the same and equal to a 45, a standard middle age of a human.

12.1 Scenario 1) "Small Meeting"

Time: mid-June

Participants:

- Monterey CA, USA
- Zutphen, Netherlands
- Melbourne, Australia
- Shanghai, China
- Hong Kong (SAR), China
- Moscow, Russia

Screenshot (Figure 8):

12.2 Scenario 2) "Big meeting"

Time: January

Participants:

- Boston MA, USA (2 people)

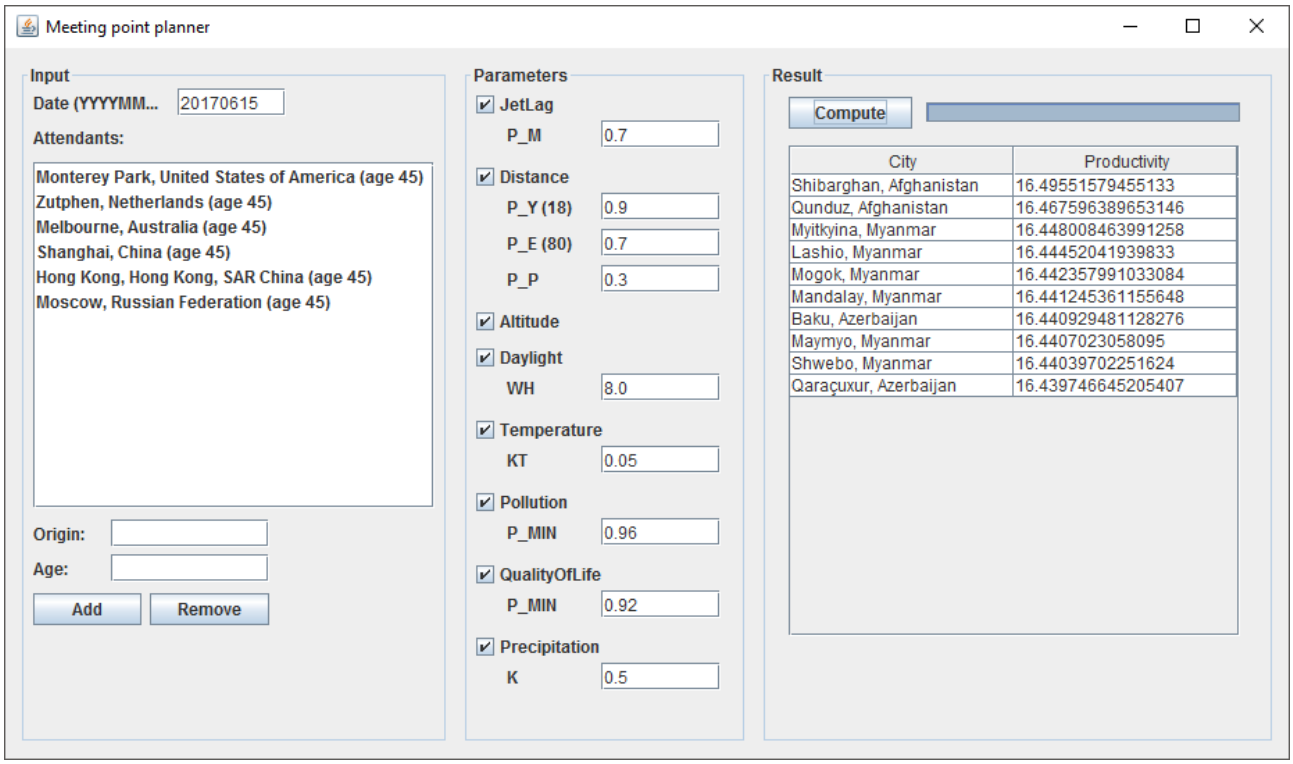

Figure 8: Results for scenario 1

- Singapore
- Beijing, China
- Hong Kong (SAR), China (2 people)
- Moscow, Russia
- Utrecht, Netherlands
- Warsaw, Poland
- Copenhagen, Denmark
- Melbourne, Australia

Screenshot (Figure 9):

It is important to note that the result highly depends on the parameters that were chosen by us and would normally be set to reflect the basic behavior of people.

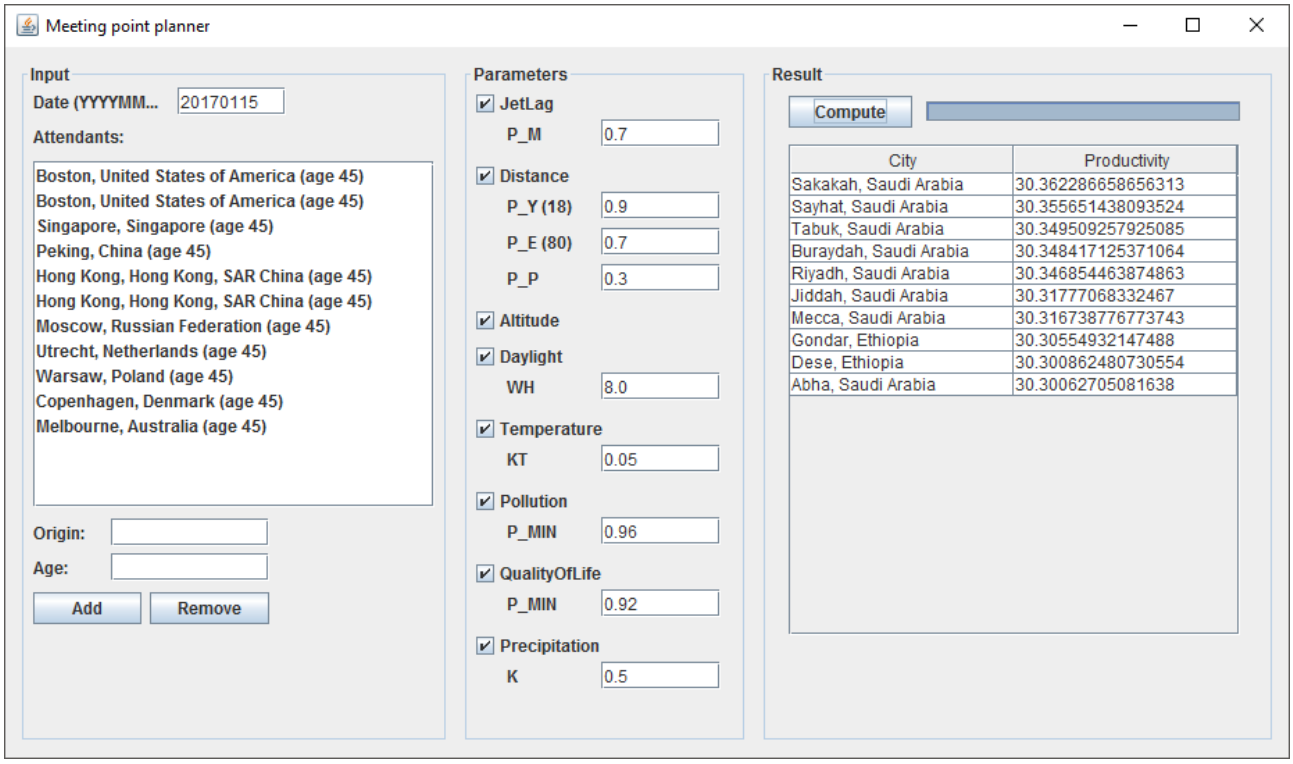

Figure 9: Results for scenario 2

References

[1] Ron Chepesiuk. "Missing the dark: health effects of light pollution." In: *Environmental health perspectives* 117.1 (Jan. 2009), A20–7. ISSN: 0091-6765. URL: http://www.ncbi.nlm.nih.gov/pubmed/19165374%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC2627884.

[2] Raj Chetty et al. "Feeling the Heat: Temperature, Physiology & the Wealth of Nations". In: (2013). URL: http://www.nber.org/papers/w19725.

[3] Kwangwook Cho et al. "Chronic Jet Lag Produces Cognitive Deficits". In: (). URL: https://pdfs.semanticscholar.org/bd7d/3455a8f033693bbc8e7b9851ded3e9325f11.pdf.

[4] Soon-Cheol Chung et al. "Physiological mechanism underlying the improvement in visuospatial performance due to 30% oxygen inhalation". In: *Applied Ergonomics* 39.2 (2008), pp. 166–170. ISSN: 00036870. DOI: 10.1016/j.apergo.2007.05.008.

[5] Soon-Cheol Chung et al. "The effect of 30% oxygen on visuospatial performance and brain activation: An fMRI study". In: *Brain and Cognition* 56.3 (2004), pp. 279–285. ISSN: 02782626. DOI: 10.1016/j.bandc.2004.07.005.

[6] Heschong Mahone Group. *Daylighting in Schools, Additional Analysis*. 2002.

[7] Jiri Richter. *How to fight the jet lag – Sleep as Android*. 2016. URL: http://sleep.urbandroid.org/how-to-fight-the-jet-lag/ (visited on 04/13/2017).

[8] Jooa Julia Lee, Francesca Gino, and Bradley R Staats. "Rainmakers: Why Bad Weather Means Good Productivity". In: ().

[9] Björn Lemmer et al. "JET LAG IN ATHLETES AFTER EASTWARD AND WESTWARD TIME-ZONE TRANSITION". In: *Chronobiology International* 19.4 (Jan. 2002), pp. 743–764. ISSN: 0742-0528. DOI: 10.1081/CBI-120005391. URL: http://www.tandfonline.com/doi/full/10.1081/CBI-120005391.

[10] Matthew D Muller et al. "Acute cold exposure and cognitive function: evidence for sustained impairment." In: *Ergonomics* 55.7 (2012), pp. 792–8. ISSN: 1366-5847. DOI: 10.1080/00140139.2012.665497. URL: http://www.ncbi.nlm.nih.gov/pubmed/22506538%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC3375336.

[11] Institute of Medicine (US) Committee on Military Nutrition Research, Bernadette M. Marriott, and Sydne J. Carlson. "The Effect of Altitude on Cognitive Performance and Mood States". In: (1996). URL: https://www.ncbi.nlm.nih.gov/books/NBK232882/.

[12] Robert L. Sack. "Jet Lag". In: *New England Journal of Medicine* 362.5 (Feb. 2010), pp. 440–447. ISSN: 0028-4793. DOI: 10.1056/NEJMcp0909838. URL: http://www.nejm.org/doi/abs/10.1056/NEJMcp0909838.

[13] Andrew B Scholey et al. "Cognitive Performance, Hyperoxia, and Heart Rate Following Oxygen Administration in Healthy Young Adults". In: *Physiology & Behavior* 67.5 (1999), pp. 783–789. ISSN: 00319384. DOI: 10.1016/S0031-9384(99)00183-3. URL: http://www.sciencedirect.com/science/article/pii/S0031938499001833.

[14] S. A Stansfeld and Mark P Matheson. "Noise pollution: non-auditory effects on health". In: *British Medical Bulletin* 68.1 (Dec. 2003), pp. 243–257. ISSN: 1471-8391. DOI: 10.1093/bmb/ldg033. URL: https://academic.oup.com/bmb/article-lookup/doi/10.1093/bmb/ldg033.

[15] Lee Taylor et al. "The Impact of Different Environmental Conditions on Cognitive Function: A Focused Review." In: *Frontiers in physiology* 6 (2015), p. 372. DOI: 10.3389/fphys.2015.00372. URL: http://www.ncbi.nlm.nih.gov/pubmed/26779029%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC4701920.

[16] Tracie White. *Study finds possible new jet-lag treatment: Exposure to flashing light | News Center | Stanford Medicine*. 2016. URL: https://med.stanford.edu/news/all-news/2016/02/study-finds-possible-new-jet-lag-treatment.html (visited on 04/13/2017).

Appendix A The code

Listing of the following files:

.gitignore
pom.xml
src/main/java/com/jjurm/projects/mpp/algorithm/Algorithm.java
src/main/java/com/jjurm/projects/mpp/algorithm/DiscreteAlgorithm.java
src/main/java/com/jjurm/projects/mpp/api/ApiManager.java
src/main/java/com/jjurm/projects/mpp/api/BingApi.java
src/main/java/com/jjurm/projects/mpp/db/CitiesImporter.java
src/main/java/com/jjurm/projects/mpp/db/DatabaseManager.java
src/main/java/com/jjurm/projects/mpp/db/PlaceFinder.java
src/main/java/com/jjurm/projects/mpp/db/QueryCache.java
src/main/java/com/jjurm/projects/mpp/db/SingleQueryCache.java
src/main/java/com/jjurm/projects/mpp/map/AltitudeMap.java
src/main/java/com/jjurm/projects/mpp/map/DaylightMap.java
src/main/java/com/jjurm/projects/mpp/map/DistanceMap.java
src/main/java/com/jjurm/projects/mpp/map/JetLagMap.java
src/main/java/com/jjurm/projects/mpp/map/PollutionMap.java
src/main/java/com/jjurm/projects/mpp/map/PrecipitationMap.java
src/main/java/com/jjurm/projects/mpp/map/ProductivityMap.java
src/main/java/com/jjurm/projects/mpp/map/ProductivityMapsFactory.java
src/main/java/com/jjurm/projects/mpp/map/QualityOfLifeMap.java
src/main/java/com/jjurm/projects/mpp/map/TemperatureMap.java
src/main/java/com/jjurm/projects/mpp/model/Attendant.java
src/main/java/com/jjurm/projects/mpp/model/Parameters.java
src/main/java/com/jjurm/projects/mpp/model/Place.java
src/main/java/com/jjurm/projects/mpp/system/Application.java
src/main/java/com/jjurm/projects/mpp/util/Holder.java

.gitignore

```
# === Java ===
*.class

# Mobile Tools for Java (J2ME)
.mtj.tmp/

# Package Files
*.jar
*.war
*.ear

# virtual machine crash logs, see http://www.java.com/en/download/help/error_hotspot.xml
hs_err_pid*


# === Maven ===
target/
pom.xml.tag
pom.xml.releaseBackup
pom.xml.versionsBackup
pom.xml.next
release.properties
dependency-reduced-pom.xml
buildNumber.properties
.mvn/timing.properties


# === Eclipse ===
*.pydevproject
.metadata
.gradle
bin/
tmp/
*.tmp
*.bak
*.swp
*~.nib
local.properties
.settings/
.loadpath

# Eclipse Core
.project

# External tool builders
.externalToolBuilders/

# Locally stored "Eclipse launch configurations"
*.launch
```

```
# CDT-specific
.cproject

# JDT-specific (Eclipse Java Development Tools)
.classpath

# Java annotation processor (APT)
.factorypath

# PDT-specific
.buildpath

# sbteclipse plugin
.target

# TeXlipse plugin
.texlipse

# STS (Spring Tool Suite)
.springBeans


# === Project-related ===
*ApiKey.txt
*.csv
```

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.jjurm.projects</groupId>
  <artifactId>meeting-point-planner</artifactId>
  <version>0.1-SNAPSHOT</version>

  <name>meeting-point-planner</name>
  <url>https://github.com/jjurm/meeting-point-planner</url>
  <packaging>jar</packaging>

  <properties>
    <jdk.version>1.8</jdk.version>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <java.main.class>com.jjurm.projects.mpp.system.Application</java.main.class>
  </properties>

  <build>
    <plugins>
      <!-- Set a JDK compiler level -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.6.1</version>
        <configuration>
          <source>${jdk.version}</source>
          <target>${jdk.version}</target>
        </configuration>
      </plugin>

      <!-- Configuration of the 'exec' plugin -->
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>exec-maven-plugin</artifactId>
        <version>1.5.0</version>
        <configuration>
          <mainClass>${java.main.class}</mainClass>
        </configuration>
      </plugin>
    </plugins>
  </build>

  <dependencies>
    <!-- https://mvnrepository.com/artifact/junit/junit -->
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/com.google.maps/google-maps-services -->
    <dependency>
      <groupId>com.google.maps</groupId>
      <artifactId>google-maps-services</artifactId>
      <version>0.1.19</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>6.0.6</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/com.peertopark.java/geocalc -->
    <dependency>
      <groupId>com.peertopark.java</groupId>
      <artifactId>geocalc</artifactId>
      <version>1.0.3</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/net.snaq/dbpool -->
    <dependency>
```

```
      <groupId>net.snaq</groupId>
      <artifactId>dbpool</artifactId>
      <version>7.0.1</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/commons-io/commons-io -->
    <dependency>
      <groupId>commons-io</groupId>
      <artifactId>commons-io</artifactId>
      <version>2.5</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.mybatis/mybatis -->
    <dependency>
      <groupId>org.mybatis</groupId>
      <artifactId>mybatis</artifactId>
      <version>3.4.3</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.json/json -->
    <dependency>
      <groupId>org.json</groupId>
      <artifactId>json</artifactId>
      <version>20160810</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.commons/commons-lang3 -->
    <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-lang3</artifactId>
      <version>3.5</version>
    </dependency>
  </dependencies>
```

src/main/java/com/jjurm/projects/mpp/algorithm/Algorithm.java

```
package com.jjurm.projects.mpp.algorithm;

import java.util.Date;
import java.util.TreeSet;

import com.jjurm.projects.mpp.map.ProductivityMapsFactory;
import com.jjurm.projects.mpp.model.Attendant;
import com.jjurm.projects.mpp.model.Place;

public abstract class Algorithm {

  protected int resultCount;

  public Algorithm(int resultCount) {
    this.resultCount = resultCount;
  }

  public abstract TreeSet<Result> find(Date date, Attendant[] attendants,
      ProductivityMapsFactory mapsFactory);

  public static class Result {

    public static Object[] tableColumns = {"City", "Productivity"};

    private double productivitySum;
    private Place destination;

    public Result(double productivitySum, Place destination) {
      super();
      this.productivitySum = productivitySum;
      this.destination = destination;
    }

    public Object[] getTableRow() {
      return new Object[] {destination, productivitySum};
    }

    public double getProductivitySum() {
      return productivitySum;
    }

    public Place getDestination() {
      return destination;
    }

    @Override
    public String toString() {
      return destination + " (" + productivitySum + ")";
    }

  }

}
```

src/main/java/com/jjurm/projects/mpp/algorithm/DiscreteAlgorithm.java

```
package com.jjurm.projects.mpp.algorithm;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Comparator;
```

```java
import java.util.Date;
import java.util.TreeSet;
import java.util.function.Consumer;

import com.jjurm.projects.mpp.db.DatabaseManager;
import com.jjurm.projects.mpp.db.PlaceFinder;
import com.jjurm.projects.mpp.map.ProductivityMap;
import com.jjurm.projects.mpp.map.ProductivityMapsFactory;
import com.jjurm.projects.mpp.model.Attendant;
import com.jjurm.projects.mpp.model.Place;

public class DiscreteAlgorithm extends Algorithm {

  protected Consumer<Double> progressUpdater;

  public DiscreteAlgorithm(int resultCount, Consumer<Double> progressUpdater) {
    super(resultCount);
    this.progressUpdater = progressUpdater;
  }

  protected void updateProgress(double progress) {
    if (progressUpdater != null) {
      progressUpdater.accept(progress);
    }
  }

  @Override
  public TreeSet<Algorithm.Result> find(Date date, Attendant[] attendants,
      ProductivityMapsFactory mapsFactory) {

    updateProgress(0);

    ProductivityMap[][] maps = new ProductivityMap[attendants.length][];

    for (int i = 0; i < attendants.length; i++) {
      maps[i] = mapsFactory.produce(date, attendants[i]);
    }

    TreeSet<Algorithm.Result> results =
        new TreeSet<Algorithm.Result>(new Comparator<Algorithm.Result>() {
          @Override
          public int compare(Result o1, Result o2) {
            if (o2.getProductivitySum() == o1.getProductivitySum())
              return o1.getDestination().getId() - o2.getDestination().getId();
            else
              return o1.getProductivitySum() < o2.getProductivitySum() ? 1 : -1;
          }
        });

    String query = PlaceFinder.QUERY_BASE;
    int rowCount, row = 0;
    try (Connection conn = DatabaseManager.getConnection();
        Statement stmt = conn.createStatement();
        ResultSet result = stmt.executeQuery(query);) {

      result.last();
      rowCount = result.getRow();
      result.beforeFirst();

      double sum, productivity;

      while (result.next()) {
        Place destination = PlaceFinder.fromResultSet(result);

        sum = 0;
        for (int day = 1; day <= 3; day++) {
          for (int attendant = 0; attendant < attendants.length; attendant++) {
            productivity = 1;
            for (int map = 0; map < maps[attendant].length; map++) {
              productivity *= maps[attendant][map].calculateProductivity(destination, day);
            }
            sum += productivity;
          }
        }

        // System.out.println(destination + " " + sum);
        Algorithm.Result r = new Algorithm.Result(sum, destination);
        if (results.size() < resultCount) {
          results.add(r);
        } else if (sum > results.last().getProductivitySum()) {
          results.pollLast();
          results.add(r);
        }

        row++;
        updateProgress(((double) row) / rowCount);

      }

    } catch (SQLException e) {
      e.printStackTrace();
    }

    return results;
  }

}
```

```java
package com.jjurm.projects.mpp.api;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

import org.apache.commons.io.FileUtils;

import com.google.maps.GeoApiContext;

public class ApiManager {

  static GeoApiContext context = null;

  public static GeoApiContext getContext() {
    if (context == null) {
      try (BufferedReader br = new BufferedReader(new FileReader("ApiKey.txt"))) {
        String line = br.readLine();
        context = new GeoApiContext().setApiKey(line);
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
    return context;
  }

  static String bingApiKey = null;

  public static String getBingApiKey() {
    if (bingApiKey == null) {
      try {
        bingApiKey = FileUtils.readFileToString(new File("BingApiKey.txt"), "UTF-8");
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
    return bingApiKey;
  }

}
```

src/main/java/com/jjurm/projects/mpp/api/BingApi.java

```java
package com.jjurm.projects.mpp.api;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.Reader;
import java.net.URL;
import java.nio.charset.Charset;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

public class BingApi {

  private static final String ALPHABET =
      "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789_-";

  private static final String BASE_URL =
      "http://dev.virtualearth.net/REST/v1/Elevation/List?key=" + ApiManager.getBingApiKey();

  public static int[] fetchElevation(double[][] points, int count) {
    // String url = BASE_URL + "&points=" + compressPoints(points, count);
    String url = BASE_URL + "&points=" + concatenatePoints(points, count);
    int[] result = new int[count];
    try {
      JSONObject json = readJsonFromUrl(url);
      JSONArray elevations = json.getJSONArray("resourceSets").getJSONObject(0)
          .getJSONArray("resources").getJSONObject(0).getJSONArray("elevations");
      for (int i = 0; i < count; i++) {
        result[i] = elevations.getInt(i);
      }
      return result;
    } catch (JSONException | IOException e) {
      e.printStackTrace();
    }
    return null;
  }

  public static JSONObject readJsonFromUrl(String url) throws IOException, JSONException {
    InputStream is = new URL(url).openStream();
    try {
      BufferedReader rd = new BufferedReader(new InputStreamReader(is, Charset.forName("UTF-8")));
      String jsonText = readAll(rd);
      JSONObject json = new JSONObject(jsonText);
      return json;
    } finally {
      is.close();
    }
  }

  private static String readAll(Reader rd) throws IOException {
    StringBuilder sb = new StringBuilder();
    int cp;
```

```
    while ((cp = rd.read()) != -1) {
      sb.append((char) cp);
    }
    return sb.toString();
  }

  public static String concatenatePoints(double[][] points, int count) {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < count; i++) {
      sb.append(points[i][0] + "," + points[i][1]);
      if (i + 1 < count) {
        sb.append(",");
      }
    }
    return sb.toString();
  }

  public static String compressPoints(double[][] points, int count) {
    int latitude = 0, longitude = 0, dx, dy, newLatitude, newLongitude, index, rem;
    StringBuilder sb = new StringBuilder();
    // double l;

    for (int point = 0; point < count; point++) {

      // step 2
      newLatitude = (int) Math.round(points[point][0] * 100000);
      newLongitude = (int) Math.round(points[point][1] * 100000);

      // step 3
      dy = newLatitude - latitude;
      dx = newLongitude - longitude;
      latitude = newLatitude;
      longitude = newLongitude;

      // step 4 and 5
      dy = (dy << 1) ^ (dy >> 31);
      dx = (dx << 1) ^ (dx >> 31);

      // step 6
      index = ((dy + dx) * (dy + dx + 1) / 2) + dy;

      while (index > 0) {

        // step 7
        rem = index & 31;
        index = (index - rem) / 32;

        // step 8
        if (index > 0)
          rem += 32;

        // step 9
        sb.append(ALPHABET.charAt(rem));
      }
    }

    // step 10
    return sb.toString();
  }

}
```

src/main/java/com/jjurm/projects/mpp/db/CitiesImporter.java

```
package com.jjurm.projects.mpp.db;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Types;
import java.util.TimeZone;

import org.apache.ibatis.jdbc.ScriptRunner;

import com.google.maps.ElevationApi;
import com.google.maps.GeoApiContext;
import com.google.maps.PendingResult;
import com.google.maps.TimeZoneApi;
import com.google.maps.errors.ApiException;
import com.google.maps.errors.ZeroResultsException;
import com.google.maps.model.ElevationResult;
import com.google.maps.model.LatLng;
import com.jjurm.projects.mpp.api.ApiManager;
import com.jjurm.projects.mpp.api.BingApi;

public class CitiesImporter {

  static String filename = "cities.csv";
  static int ignoreLines = 1;

  public static void run() {
    DatabaseManager.init();
    // importCities();
    // fetchTimezones();
```

```
    fetchElevations();
    DatabaseManager.release();
  }

  public static void importCities() {

    String insert2Statement =
        "INSERT INTO `bigcities`(`country`, `city`, `accent`, `region`, `population`, `lat`, `lon`) VALUES (?,
        ↪  ?, ?, ?, ?, ?, ?)";

    try (BufferedReader brCsv = new BufferedReader(new FileReader(filename));
        BufferedReader brCitiesSql = new BufferedReader(new FileReader("sql/cities.sql"));
        Connection conn = DatabaseManager.getConnection();) {

      ScriptRunner sr = new ScriptRunner(conn);
      sr.runScript(brCitiesSql);

      conn.setAutoCommit(false);

      try (PreparedStatement stmt2 = conn.prepareStatement(insert2Statement)) {

        for (int i = 0; i < ignoreLines; i++) {
          brCsv.readLine();
        }
        String line;
        String[] parts;
        Integer population;
        while ((line = brCsv.readLine()) != null) {
          parts = line.split(",");

          if (parts[4].length() > 0) {
            population = Integer.parseInt(parts[4]);
          } else {
            population = null;
          }

          if (population != null && population > 50000) {
            stmt2.setString(1, parts[0]);
            stmt2.setString(2, parts[1]);
            stmt2.setString(3, parts[2]);
            stmt2.setString(4, parts[3]);
            stmt2.setInt(5, population);
            stmt2.setDouble(6, Double.parseDouble(parts[5]));
            stmt2.setDouble(7, Double.parseDouble(parts[6]));
            stmt2.executeUpdate();
          }
        }
        conn.commit();

      }

    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    } catch (SQLException e) {
      e.printStackTrace();
    }
  }

  public static void fetchTimezones() {
    try (Connection conn = DatabaseManager.getConnection()) {
      conn.setAutoCommit(false);

      try (
          PreparedStatement stmtSelect =
              conn.prepareStatement("SELECT id, lat, lon FROM bigcities");
          PreparedStatement stmtUpdate =
              conn.prepareStatement("UPDATE bigcities SET tz_id = ?, alt = ? WHERE id = ?");
          ResultSet result = stmtSelect.executeQuery()) {

        ApiFunctionProcessor<TimeZone> pTimeZone = new ApiFunctionProcessor<>(
            TimeZoneApi::getTimeZone, tz -> stmtUpdate.setString(1, tz.getID()),
            () -> stmtUpdate.setNull(1, Types.VARCHAR));
        ApiFunctionProcessor<ElevationResult> pElevation =
            new ApiFunctionProcessor<ElevationResult>(ElevationApi::getByPoint,
                er -> stmtUpdate.setDouble(2, er.elevation),
                () -> stmtUpdate.setNull(1, Types.DOUBLE));

        int i = 0;
        while (result.next()) {
          if (i % 100 == 0) {
            conn.commit();
            System.out.println(i);
          }
          LatLng point = new LatLng(result.getDouble("lat"), result.getDouble("lon"));
          stmtUpdate.setInt(3, result.getInt("id"));
          pTimeZone.process(point);
          pElevation.process(point);
          stmtUpdate.executeUpdate();
          i++;
        }

        conn.commit();
      }
    } catch (SQLException e) {
      e.printStackTrace();
    } catch (ApiException e) {
      e.printStackTrace();
```

```
    } catch (InterruptedException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }

  }

  public static void fetchElevations() {
    try (Connection conn = DatabaseManager.getConnection()) {
      conn.setAutoCommit(false);

      try (
          PreparedStatement stmtSelect =
              conn.prepareStatement("SELECT id, lat, lon FROM bigcities");
          PreparedStatement stmtUpdate =
              conn.prepareStatement("UPDATE bigcities SET alt = ? WHERE id = ?");
          ResultSet result = stmtSelect.executeQuery()) {

        int i = 0;
        int count = 0;
        int[] ids = new int[100];
        double[][] points = new double[100][2];
        while (result.next() && i < 150) {
          if (count == 40) {
            int[] elevations = BingApi.fetchElevation(points, count);
            for (int j = 0; j < count; j++) {
              stmtUpdate.setInt(2, ids[j]);
              stmtUpdate.setInt(1, elevations[j]);
              stmtUpdate.executeUpdate();
            }
            count = 0;

            conn.commit();
            System.out.println(i);
          }
          ids[count] = result.getInt(1);
          points[count][0] = result.getDouble(2);
          points[count][1] = result.getDouble(3);
          i++;
          count++;
        }

        int[] elevations = BingApi.fetchElevation(points, count);
        for (int j = 0; j < count; j++) {
          stmtUpdate.setInt(2, ids[0]);
          stmtUpdate.setInt(1, elevations[j]);
          stmtUpdate.executeUpdate();
        }

        conn.commit();
      }
    } catch (SQLException e) {
      e.printStackTrace();
    }

  }

  private static class ApiFunctionProcessor<T> {

    private ApiFunction<T> function;
    private ResultConsumer<T> onSuccess;
    private NoResultAction action;

    public ApiFunctionProcessor(ApiFunction<T> function, ResultConsumer<T> onSuccess,
        NoResultAction action) {
      this.function = function;
      this.onSuccess = onSuccess;
      this.action = action;
    }

    public void process(LatLng point)
        throws ApiException, InterruptedException, IOException, SQLException {
      PendingResult<T> result = function.query(ApiManager.getContext(), point);
      try {
        T r = result.await();
        onSuccess.accept(r);
      } catch (ZeroResultsException e) {
        action.run();
      }
    }

    private static interface ApiFunction<T> {
      public PendingResult<T> query(GeoApiContext context, LatLng point);
    }

    private static interface ResultConsumer<T> {
      public void accept(T r) throws SQLException;
    }

    private static interface NoResultAction {
      public void run() throws SQLException;
    }

  }

}
```

src/main/java/com/jjurm/projects/mpp/db/DatabaseManager.java

```java
package com.jjurm.projects.mpp.db;

import java.sql.Connection;
import java.sql.SQLException;

import snaq.db.ConnectionPool;

public class DatabaseManager {

  private static final String URL =
      "jdbc:mysql://localhost/immc?user=root&useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=fal

  private static ConnectionPool pool;

  public static void init() {
    pool = new ConnectionPool("pool", 1, 3, 3, 0, URL, null);
  }

  public static void release() {
    pool.release();
  }

  public static Connection getConnection() throws SQLException {
    return pool.getConnection();
  }

}
```

src/main/java/com/jjurm/projects/mpp/db/PlaceFinder.java

```java
package com.jjurm.projects.mpp.db;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.TimeZone;

import com.jjurm.projects.mpp.map.PollutionMap;
import com.jjurm.projects.mpp.map.QualityOfLifeMap;
import com.jjurm.projects.mpp.model.Place;
import com.peertopark.java.geocalc.DegreeCoordinate;
import com.peertopark.java.geocalc.Point;

public class PlaceFinder {

  public static final String QUERY_BASE =
      "SELECT bigcities.id as id, bigcities.country as country, accent, population, lat, lon, tz_id, alt,
      countries.name, temperature.*, precipitation.*, pollution.pollution as pollution, qol.qol as qol "
          + "FROM bigcities LEFT JOIN countries ON (UPPER(bigcities.country) = countries.alpha2) "
          + "LEFT JOIN temperature ON (countries.alpha3 = temperature.country) LEFT JOIN precipitation ON
          (countries.alpha3 = precipitation.country) LEFT JOIN pollution ON (countries.name =
          pollution.country) LEFT JOIN qol ON (countries.name = qol.country) "
          + "WHERE tz_id is not null and bigcities.ign = 0";

  public static Place id(int id) throws NotFoundException, SQLException {
    return query("id = " + id);
  }

  public static Place city(String city) throws NotFoundException, SQLException {
    return query("city like '%" + city + "%'");
  }

  public static Place accent(String accent) throws NotFoundException, SQLException {
    return query("accent like '%" + accent + "%'");
  }

  public static Place fromResultSet(ResultSet result) throws SQLException {
    int id = result.getInt("id");
    String name = result.getString("accent");
    String country_name = result.getString("countries.name");
    Point point = new Point(new DegreeCoordinate(result.getDouble("lat")),
        new DegreeCoordinate(result.getDouble("lon")));
    int population = result.getInt("population");
    TimeZone timezone = TimeZone.getTimeZone(result.getString("tz_id"));
    int altitude = result.getInt("alt");

    double[] temperature = new double[13];
    temperature[0] = result.getDouble("tjan");
    temperature[1] = result.getDouble("tfeb");
    temperature[2] = result.getDouble("tmar");
    temperature[3] = result.getDouble("tapr");
    temperature[4] = result.getDouble("tmay");
    temperature[5] = result.getDouble("tjun");
    temperature[6] = result.getDouble("tjul");
    temperature[7] = result.getDouble("taug");
    temperature[8] = result.getDouble("tsep");
    temperature[9] = result.getDouble("toct");
    temperature[10] = result.getDouble("tnov");
    temperature[11] = result.getDouble("tdec");
    temperature[12] = result.getDouble("tannual");
    double[] precipitation = new double[13];
    precipitation[0] = result.getDouble("pjan");
    precipitation[1] = result.getDouble("pfeb");
```

```
62      precipitation[2] = result.getDouble("pmar");
63      precipitation[3] = result.getDouble("papr");
64      precipitation[4] = result.getDouble("pmay");
65      precipitation[5] = result.getDouble("pjun");
66      precipitation[6] = result.getDouble("pjul");
67      precipitation[7] = result.getDouble("paug");
68      precipitation[8] = result.getDouble("psep");
69      precipitation[9] = result.getDouble("poct");
70      precipitation[10] = result.getDouble("pnov");
71      precipitation[11] = result.getDouble("pdec");
72      precipitation[12] = result.getDouble("pannual");
73      double pollution = result.getDouble("pollution");
74      if (result.wasNull()) {
75        pollution = PollutionMap.getMeanPollution();
76      }
77      double qol = result.getDouble("qol");
78      if (result.wasNull()) {
79        qol = QualityOfLifeMap.getMeanQOL();
80      }
81
82      Place place = new Place(id, name, country_name, point, population, timezone, altitude,
83          temperature, precipitation, pollution, qol);
84      return place;
85    }
86
87    protected static Place query(String condition) throws NotFoundException, SQLException {
88      String query = QUERY_BASE + " AND " + condition + " ORDER BY population DESC LIMIT 1";
89      try (Connection conn = DatabaseManager.getConnection();
90          Statement stmt = conn.createStatement();
91          ResultSet result = stmt.executeQuery(query)) {
92        if (result.first()) {
93          return fromResultSet(result);
94        } else {
95          throw new NotFoundException("Place not found (" + condition + ")");
96        }
97      }
98    }
99
100   public static class NotFoundException extends Exception {
101     private static final long serialVersionUID = 1L;
102
103     public NotFoundException(String s) {
104       super(s);
105     }
106   }
107
108 }
```

src/main/java/com/jjurm/projects/mpp/db/QueryCache.java

```
1   package com.jjurm.projects.mpp.db;
2
3   import java.sql.Connection;
4   import java.sql.ResultSet;
5   import java.sql.SQLException;
6   import java.sql.Statement;
7   import java.util.HashMap;
8
9   public class QueryCache<K, V> {
10
11    private QueryBuilder<K> queryBuilder;
12    private ResultExtractor<V> resultExtractor;
13
14    private HashMap<K, V> cache = new HashMap<K, V>();
15
16    public QueryCache(QueryBuilder<K> queryBuilder, ResultExtractor<V> resultExtractor) {
17      this.queryBuilder = queryBuilder;
18      this.resultExtractor = resultExtractor;
19    }
20
21    public V getValue(K key) {
22
23      if (cache.containsKey(key)) {
24        return cache.get(key);
25      }
26
27      String query = queryBuilder.buildFrom(key);
28      try (Connection conn = DatabaseManager.getConnection();
29          Statement stmt = conn.createStatement();
30          ResultSet result = stmt.executeQuery(query);) {
31        if (result.first()) {
32          V value = resultExtractor.getFrom(result);
33          cache.put(key, value);
34          return value;
35        } else {
36          return null;
37        }
38      } catch (SQLException e) {
39        e.printStackTrace();
40        return null;
41      }
42    }
43
44    public static interface QueryBuilder<K> {
45      public String buildFrom(K key);
46    }
47
48    public static interface ResultExtractor<V> {
```

```
49      public V getFrom(ResultSet result) throws SQLException;
50    }
51
52  }
```

src/main/java/com/jjurm/projects/mpp/db/SingleQueryCache.java

```
1   package com.jjurm.projects.mpp.db;
2
3   import java.sql.Connection;
4   import java.sql.ResultSet;
5   import java.sql.SQLException;
6   import java.sql.Statement;
7
8   public class SingleQueryCache<V> {
9
10    private String query;
11    private ResultExtractor<V> resultExtractor;
12
13    private V result = null;
14
15    public SingleQueryCache(String query, ResultExtractor<V> resultExtractor) {
16      this.query = query;
17      this.resultExtractor = resultExtractor;
18    }
19
20    public V getValue() {
21
22      if (result != null) {
23        return result;
24      }
25
26      try (Connection conn = DatabaseManager.getConnection();
27          Statement stmt = conn.createStatement();
28          ResultSet resultSet = stmt.executeQuery(query);) {
29        if (resultSet.first()) {
30          V value = resultExtractor.getFrom(resultSet);
31          result = value;
32          return value;
33        } else {
34          return null;
35        }
36      } catch (SQLException e) {
37        e.printStackTrace();
38        return null;
39      }
40    }
41
42    public static interface ResultExtractor<V> {
43      public V getFrom(ResultSet result) throws SQLException;
44    }
45
46  }
```

src/main/java/com/jjurm/projects/mpp/map/AltitudeMap.java

```
1   package com.jjurm.projects.mpp.map;
2
3   import java.util.Date;
4
5   import com.jjurm.projects.mpp.model.Attendant;
6   import com.jjurm.projects.mpp.model.Parameters.ParametersList;
7   import com.jjurm.projects.mpp.model.Place;
8
9   public class AltitudeMap extends ProductivityMap {
10
11    public static final double T0 = 288.15;
12    public static final double L = 0.0065;
13    public static final double p0 = 101325;
14    public static final double g = 9.80665;
15    public static final double M = 0.0289644;
16    public static final double R = 8.31447;
17
18    public static final double a = 3.995;
19    public static final double b = 0.7576;
20    public static final double c = 14.05537;
21
22    public AltitudeMap(ParametersList parameters, Date date, Attendant attendant) {
23      super(parameters, date, attendant);
24    }
25
26    @Override
27    public double calculateProductivity(Place destination, int day) {
28      int h = destination.getAltitude();
29
30      double g0 = Math.log(M * p0 / (R * T0) - b) / Math.log(a) + c;
31      double gh =
32          Math.log((M * p0 * Math.pow(1 - L * h / T0, g * M / (R * L))) / (R * (T0 - L * h)) - b)
33              / Math.log(a) + c;
34
35      return gh / g0;
36    }
37
38  }
```

src/main/java/com/jjurm/projects/mpp/map/DaylightMap.java

```java
package com.jjurm.projects.mpp.map;

import java.util.Calendar;
import java.util.Date;

import com.jjurm.projects.mpp.db.QueryCache;
import com.jjurm.projects.mpp.model.Attendant;
import com.jjurm.projects.mpp.model.Parameters.ParametersList;
import com.jjurm.projects.mpp.model.Place;

public class DaylightMap extends ProductivityMap {

  public static final double D = 100.0 / 121.0;

  public static final String PARAM_WH = "WH";

  public static QueryCache<Integer, Double> cache = new QueryCache<Integer, Double>(
      day -> "SELECT declination FROM sundeclination WHERE `day` = " + day, rs -> rs.getDouble(1));

  public DaylightMap(ParametersList parameters, Date date, Attendant attendant) {
    super(parameters, date, attendant);
  }

  @Override
  public double calculateProductivity(Place destination, int day) {
    double phi = destination.getPoint().getLatitude() * Math.PI / 180;

    Calendar calendar = Calendar.getInstance();
    calendar.setTime(date);
    int dayOfYear = (calendar.get(Calendar.DAY_OF_YEAR) + day) % 366;
    double declination = cache.getValue(dayOfYear) * Math.PI / 180;

    double h = Math.acos(-Math.tan(phi) * Math.tan(declination)) / (2 * Math.PI);
    double sunrise = 24 * -h + 12;
    double sunset = 24 * h + 12;

    double wh = parameters.get(PARAM_WH);
    double RL = Math.max(0, -sunset + sunrise + wh) / wh;

    double P = RL * D + (1 - RL);
    return P;
  }

}
```

src/main/java/com/jjurm/projects/mpp/map/DistanceMap.java

```java
package com.jjurm.projects.mpp.map;

import java.util.Date;

import com.jjurm.projects.mpp.model.Attendant;
import com.jjurm.projects.mpp.model.Parameters.ParametersList;
import com.jjurm.projects.mpp.model.Place;
import com.peertopark.java.geocalc.EarthCalc;

public class DistanceMap extends ProductivityMap {

  public static final double AGE_YOUNG = 18;
  public static final double AGE_ELDERLY = 80;
  public static final double MAX_DISTANCE = 20_000_000;

  public static final String PARAM_PY = "P_Y (" + ((int) AGE_YOUNG) + ")";
  public static final String PARAM_PE = "P_E (" + ((int) AGE_ELDERLY) + ")";
  public static final String PARAM_PP = "P_P";

  private double PM;

  public DistanceMap(ParametersList parameters, Date date, Attendant attendant) {
    super(parameters, date, attendant);
    double PY = parameters.get(PARAM_PY);
    double PE = parameters.get(PARAM_PE);
    PM = PY - (attendant.getAge() - AGE_YOUNG) / (AGE_ELDERLY - AGE_YOUNG) * (PY - PE);
  }

  @Override
  public double calculateProductivity(Place destination, int day) {
    if (day > 1)
      return 1;

    double distance =
        EarthCalc.getHarvesineDistance(attendant.getOrigin().getPoint(), destination.getPoint());

    double popAddition =
        parameters.get(PARAM_PP) * Math.sqrt(destination.getPopulation()) / 1000000;

    double P = Math.exp(distance * Math.log(PM) / MAX_DISTANCE) + popAddition;
    return P;
  }
}
```

src/main/java/com/jjurm/projects/mpp/map/JetLagMap.java

```java
package com.jjurm.projects.mpp.map;

import java.util.Date;

```

```java
import com.jjurm.projects.mpp.model.Attendant;
import com.jjurm.projects.mpp.model.Parameters.ParametersList;
import com.jjurm.projects.mpp.model.Place;

public class JetLagMap extends ProductivityMap {

  public static final String PARAM_PM = "P_M";

  public static final double LE = 0.7;
  public static final double LW = 0.7;

  public JetLagMap(ParametersList parameters, Date date, Attendant attendant) {
    super(parameters, date, attendant);
  }

  @Override
  public double calculateProductivity(Place destination, int day) {
    double pm = parameters.get(PARAM_PM);

    double pw = pm * Math.sqrt(LE / LW);
    double pe = pm * Math.sqrt(LW / LE);

    int offset1 = attendant.getOrigin().getTimeZone().getOffset(date.getTime());
    int offset2 = destination.getTimeZone().getOffset(date.getTime());
    double hourDiff = (offset2 - offset1) / 3600000;

    boolean toEast = hourDiff >= 0;
    hourDiff = Math.abs(hourDiff);

    double dayCoefficient1 = toEast ? 2 : 2.5;
    double p1 = toEast ? pe : pw;
    double productivity1 =
        1 - Math.pow(Math.max(0, hourDiff - dayCoefficient1 * (day - 1)) / 12, 2) * (1 - p1);

    double dayCoefficient2 = !toEast ? 2 : 2.5;
    double p2 = !toEast ? pe : pw;
    double productivity2 =
        1 - Math.pow(Math.max(0, hourDiff - dayCoefficient2 * (day - 1)) / 12, 2) * (1 - p2);

    return Math.max(productivity1, productivity2);
  }

}
```

src/main/java/com/jjurm/projects/mpp/map/PollutionMap.java

```java
package com.jjurm.projects.mpp.map;

import java.util.Date;

import com.jjurm.projects.mpp.db.SingleQueryCache;
import com.jjurm.projects.mpp.model.Attendant;
import com.jjurm.projects.mpp.model.Parameters.ParametersList;
import com.jjurm.projects.mpp.model.Place;

public class PollutionMap extends ProductivityMap {

  public static final String PARAM_P_MIN = "P_MIN";

  public PollutionMap(ParametersList parameters, Date date, Attendant attendant) {
    super(parameters, date, attendant);
  }

  @Override
  public double calculateProductivity(Place destination, int day) {
    double x = destination.getPollution();

    double P = Math.pow(parameters.get(PARAM_P_MIN), (100 - x) / 100);
    return P;
  }

  private static SingleQueryCache<Double> cache =
      new SingleQueryCache<>("SELECT AVG(pollution) as avg FROM pollution", r -> r.getDouble(1));

  public static double getMeanPollution() {
    return cache.getValue();
  }

}
```

src/main/java/com/jjurm/projects/mpp/map/PrecipitationMap.java

```java
package com.jjurm.projects.mpp.map;

import java.util.Calendar;
import java.util.Date;

import com.jjurm.projects.mpp.model.Attendant;
import com.jjurm.projects.mpp.model.Parameters.ParametersList;
import com.jjurm.projects.mpp.model.Place;

public class PrecipitationMap extends ProductivityMap {

  public static final String PARAM_K = "K";

  public static final double P_DECREASE = 0.01363;

```

```
  public PrecipitationMap(ParametersList parameters, Date date, Attendant attendant) {
    super(parameters, date, attendant);
  }

  @Override
  public double calculateProductivity(Place destination, int day) {
    Calendar calendar = Calendar.getInstance();
    calendar.setTime(date);
    calendar.add(Calendar.DATE, day);
    int month = calendar.get(Calendar.MONTH);

    double x = destination.getPrecipitation(month);
    double K = parameters.get(PARAM_K);

    double P = Math.pow(1 - P_DECREASE, -K * Math.sqrt(x) / 25.4);
    return P;
  }

}
```

src/main/java/com/jjurm/projects/mpp/map/ProductivityMap.java

```
package com.jjurm.projects.mpp.map;

import java.util.Date;

import com.jjurm.projects.mpp.model.Attendant;
import com.jjurm.projects.mpp.model.Parameters.ParametersList;
import com.jjurm.projects.mpp.model.Place;

/**
 * This abstract class represents a function that basically takes coordinates as an argument and
 * returns the adjusted productivity of an attendant after travelling to the specified location,
 * considering the factor mapped by a particular implementation.
 *
 * @author JJurM
 */
public abstract class ProductivityMap {

  protected ParametersList parameters;
  protected Date date;
  protected Attendant attendant;

  public ProductivityMap(ParametersList parameters, Date date, Attendant attendant) {
    this.parameters = parameters;
    this.date = date;
    this.attendant = attendant;
  }

  /**
   * Returns productivity given a specified destination and a day. The 0-th day is the day of
   * arrival and the meeting days are from 1 to 3 (inclusive).
   */
  public abstract double calculateProductivity(Place destination, int day);

}
```

src/main/java/com/jjurm/projects/mpp/map/ProductivityMapsFactory.java

```
package com.jjurm.projects.mpp.map;

import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.List;

import com.jjurm.projects.mpp.model.Attendant;
import com.jjurm.projects.mpp.model.Parameters;
import com.jjurm.projects.mpp.model.Parameters.ParametersList;

public class ProductivityMapsFactory {

  private static HashMap<Class<? extends ProductivityMap>, Factory> allFactories =
      new HashMap<Class<? extends ProductivityMap>, Factory>() {
        private static final long serialVersionUID = 1L;
        {
          put(AltitudeMap.class, AltitudeMap::new);
          put(DaylightMap.class, DaylightMap::new);
          put(DistanceMap.class, DistanceMap::new);
          put(JetLagMap.class, JetLagMap::new);
          put(PollutionMap.class, PollutionMap::new);
          put(PrecipitationMap.class, PrecipitationMap::new);
          put(QualityOfLifeMap.class, QualityOfLifeMap::new);
          put(TemperatureMap.class, TemperatureMap::new);
        }
      };


  List<Class<? extends ProductivityMap>> classes =
      new ArrayList<Class<? extends ProductivityMap>>();
  Parameters parameters;

  public ProductivityMapsFactory(Parameters parameters) {
    this.parameters = parameters;
  }

```

```
38    public void addFactory(Class<? extends ProductivityMap> clazz) {
39      classes.add(clazz);
40    }
41
42    public ProductivityMap[] produce(Date date, Attendant attendant) {
43      ProductivityMap[] maps = new ProductivityMap[classes.size()];
44      int i = 0;
45      for (Class<? extends ProductivityMap> clazz : classes) {
46        ParametersList list = parameters.getParametersList(clazz);
47        maps[i] = allFactories.get(clazz).construct(list, date, attendant);
48        i++;
49      }
50      return maps;
51    }
52
53    static interface Factory {
54
55      public ProductivityMap construct(ParametersList parameters, Date date, Attendant attendant);
56
57    }
58
59  }
```

src/main/java/com/jjurm/projects/mpp/map/QualityOfLifeMap.java

```
1   package com.jjurm.projects.mpp.map;
2
3   import java.util.Date;
4
5   import com.jjurm.projects.mpp.db.SingleQueryCache;
6   import com.jjurm.projects.mpp.model.Attendant;
7   import com.jjurm.projects.mpp.model.Parameters.ParametersList;
8   import com.jjurm.projects.mpp.model.Place;
9
10  public class QualityOfLifeMap extends ProductivityMap {
11
12    public static final String PARAM_P_MIN = "P_MIN";
13
14    public QualityOfLifeMap(ParametersList parameters, Date date, Attendant attendant) {
15      super(parameters, date, attendant);
16    }
17
18    @Override
19    public double calculateProductivity(Place destination, int day) {
20      double qol = destination.getQualityOfLife();
21      double max = cacheMax.getValue();
22
23      double P = Math.pow(parameters.get(PARAM_P_MIN), (max - qol) / max);
24      return P;
25    }
26
27    private static SingleQueryCache<Double> cache =
28        new SingleQueryCache<>("SELECT AVG(qol) as avg FROM qol", r -> r.getDouble(1));
29
30    private static SingleQueryCache<Double> cacheMax =
31        new SingleQueryCache<>("SELECT MAX(qol) FROM qol", r -> r.getDouble(1));
32
33    public static double getMeanQOL() {
34      return cache.getValue();
35    }
36
37  }
```

src/main/java/com/jjurm/projects/mpp/map/TemperatureMap.java

```
1   package com.jjurm.projects.mpp.map;
2
3   import java.util.Calendar;
4   import java.util.Date;
5
6   import com.jjurm.projects.mpp.model.Attendant;
7   import com.jjurm.projects.mpp.model.Parameters.ParametersList;
8   import com.jjurm.projects.mpp.model.Place;
9
10  public class TemperatureMap extends ProductivityMap {
11
12    public static final String PARAM_KT = "KT";
13
14    public static final double c1 = -42.379;
15    public static final double c2 = 2.04901523;
16    public static final double c3 = 10.14333127;
17    public static final double c4 = -0.22475541;
18    public static final double c5 = -0.683783 * Math.pow(10, -3);
19    public static final double c6 = -5.481717 * Math.pow(10, -2);
20    public static final double c7 = 1.22874 * Math.pow(10, -3);
21    public static final double c8 = 8.5280 * Math.pow(10, -4);
22    public static final double c9 = -1.99 * Math.pow(10, -6);
23
24    public TemperatureMap(ParametersList parameters, Date date, Attendant attendant) {
25      super(parameters, date, attendant);
26    }
27
28    @Override
29    public double calculateProductivity(Place destination, int day) {
30      Calendar calendar = Calendar.getInstance();
31      calendar.setTime(date);
```

```
    calendar.add(Calendar.DATE, day);
    int month = calendar.get(Calendar.MONTH);

    // dry-bulb temperature
    double tC = destination.getTemperature(month);
    // double tF = celsiusToFahrenheit(tC);

    double P = productivityInTemperature(tC);

    return P;
  }

  public double productivityInTemperature(double t) {
    return ((-1.06907 * Math.pow(10, -7) * Math.pow(t, 4) + 0.00003 * Math.pow(t, 3)
        - 0.00344 * Math.pow(t, 2) + 0.11109 * t - 0.08269) - 1) * parameters.get(PARAM_KT) + 1;
  }

  public static double celsiusToFahrenheit(double v) {
    return v * 9 / 5 + 32;
  }

  public static double fahrenheitToCelsius(double v) {
    return (v - 32) * 5 / 9;
  }

}
```

src/main/java/com/jjurm/projects/mpp/model/Attendant.java

```
package com.jjurm.projects.mpp.model;

/**
 * A class representing a person going to participate in the meeting.
 *
 * @author JJurM
 */
public class Attendant {

  private Place origin;
  private double age;

  public Attendant(Place origin, double age) {
    this.origin = origin;
    this.age = age;
  }

  public Place getOrigin() {
    return origin;
  }

  public double getAge() {
    return age;
  }

  @Override
  public String toString() {
    return origin + " (age " + ((int) age) + ")";
  }

}
```

src/main/java/com/jjurm/projects/mpp/model/Parameters.java

```
package com.jjurm.projects.mpp.model;

import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Set;
import java.util.function.Consumer;

import com.jjurm.projects.mpp.map.AltitudeMap;
import com.jjurm.projects.mpp.map.DaylightMap;
import com.jjurm.projects.mpp.map.DistanceMap;
import com.jjurm.projects.mpp.map.JetLagMap;
import com.jjurm.projects.mpp.map.PollutionMap;
import com.jjurm.projects.mpp.map.PrecipitationMap;
import com.jjurm.projects.mpp.map.ProductivityMap;
import com.jjurm.projects.mpp.map.QualityOfLifeMap;
import com.jjurm.projects.mpp.map.TemperatureMap;
import com.jjurm.projects.mpp.util.Holder;

public class Parameters {

  LinkedHashMap<Class<? extends ProductivityMap>, ParametersList> lists = new LinkedHashMap<>();

  public Parameters() {
    lists.put(JetLagMap.class, new ParametersList(true, p -> {
      p.put(JetLagMap.PARAM_PM, h(0.7));
    }));
    lists.put(DistanceMap.class, new ParametersList(true, p -> {
      p.put(DistanceMap.PARAM_PY, h(0.9));
      p.put(DistanceMap.PARAM_PE, h(0.7));
      p.put(DistanceMap.PARAM_PP, h(0.3));
    }));
    lists.put(AltitudeMap.class, new ParametersList(true, p -> {
    }));
```

```
    lists.put(DaylightMap.class, new ParametersList(true, p -> {
      p.put(DaylightMap.PARAM_WH, h(8));
    }));
    lists.put(TemperatureMap.class, new ParametersList(true, p -> {
      p.put(TemperatureMap.PARAM_KT, h(0.05));
    }));
    lists.put(PollutionMap.class, new ParametersList(true, p -> {
      p.put(PollutionMap.PARAM_P_MIN, h(0.96));
    }));
    lists.put(QualityOfLifeMap.class, new ParametersList(true, p -> {
      p.put(QualityOfLifeMap.PARAM_P_MIN, h(0.92));
    }));
    lists.put(PrecipitationMap.class, new ParametersList(true, p -> {
      p.put(PrecipitationMap.PARAM_K, h(0.5));
    }));
  }

  private static Holder<Double> h(double value) {
    return new Holder<Double>(value);
  }

  public Set<Map.Entry<Class<? extends ProductivityMap>, ParametersList>> entrySet() {
    return lists.entrySet();
  }

  public ParametersList getParametersList(Class<? extends ProductivityMap> clazz) {
    return lists.get(clazz);
  }

  public static class ParametersList {

    private boolean useThisMap;
    private LinkedHashMap<String, Holder<Double>> parameters;

    public ParametersList(boolean useThisMap,
        Consumer<LinkedHashMap<String, Holder<Double>>> paramSetter) {
      this.useThisMap = useThisMap;
      this.parameters = new LinkedHashMap<String, Holder<Double>>();
      paramSetter.accept(this.parameters);
    }

    public void setUseThisMap(boolean value) {
      this.useThisMap = value;
    }

    public boolean getUseThisMap() {
      return useThisMap;
    }

    public Set<Map.Entry<String, Holder<Double>>> entrySet() {
      return parameters.entrySet();
    }

    public Double get(String key) {
      return parameters.get(key).get();
    }

  }

}
```

src/main/java/com/jjurm/projects/mpp/model/Place.java

```
package com.jjurm.projects.mpp.model;

import java.util.TimeZone;

import com.peertopark.java.geocalc.Point;

public class Place {

  int id;
  private String name;
  private String country;
  private Point point;
  private int population;
  private TimeZone timeZone;
  private int altitude;
  private double[] temperature;
  private double[] precipitation;
  private double pollution;
  private double qol;

  public Place(int id, String name, String country, Point point, int population, TimeZone timeZone,
      int altitude, double[] temperature, double[] precipitation, double pollution, double qol) {
    this.id = id;
    this.name = name;
    this.country = country;
    this.point = point;
    this.population = population;
    this.timeZone = timeZone;
    this.altitude = altitude;
    this.temperature = temperature;
    this.precipitation = precipitation;
    this.pollution = pollution;
    this.qol = qol;
  }

  public int getId() {
```

```
    return id;
  }

  public Point getPoint() {
    return point;
  }

  public int getPopulation() {
    return population;
  }

  public TimeZone getTimeZone() {
    return timeZone;
  }

  public int getAltitude() {
    return altitude;
  }

  public double getTemperature(int month) {
    return temperature[month];
  }

  public double getPrecipitation(int month) {
    return precipitation[month];
  }

  public double getPollution() {
    return pollution;
  }

  public double getQualityOfLife() {
    return qol;
  }

  @Override
  public String toString() {
    return name + ", " + country;
  }

  @Override
  public boolean equals(Object obj) {
    if (obj == null || !(obj instanceof Place))
      return false;
    Place p = (Place) obj;
    return id == p.id;
  }

  @Override
  public int hashCode() {
    return id;
  }

}
```

src/main/java/com/jjurm/projects/mpp/system/Application.java

```
package com.jjurm.projects.mpp.system;

import java.awt.BorderLayout;
import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.SQLException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Date;
import java.util.Map;
import java.util.TreeSet;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

import javax.swing.BorderFactory;
import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JProgressBar;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.ListSelectionModel;
import javax.swing.table.DefaultTableModel;

import org.apache.commons.lang3.tuple.ImmutablePair;
import org.apache.commons.lang3.tuple.Pair;

import com.jjurm.projects.mpp.algorithm.Algorithm;
import com.jjurm.projects.mpp.algorithm.Algorithm.Result;
import com.jjurm.projects.mpp.algorithm.DiscreteAlgorithm;
import com.jjurm.projects.mpp.db.DatabaseManager;
import com.jjurm.projects.mpp.db.PlaceFinder;
import com.jjurm.projects.mpp.db.PlaceFinder.NotFoundException;
import com.jjurm.projects.mpp.map.ProductivityMap;
import com.jjurm.projects.mpp.map.ProductivityMapsFactory;
import com.jjurm.projects.mpp.model.Attendant;
```

```java
import com.jjurm.projects.mpp.model.Parameters;
import com.jjurm.projects.mpp.model.Parameters.ParametersList;
import com.jjurm.projects.mpp.model.Place;
import com.jjurm.projects.mpp.util.Holder;

public class Application {

  ExecutorService executor = Executors.newSingleThreadExecutor();

  DefaultListModel<Attendant> attendants = new DefaultListModel<Attendant>();
  DefaultTableModel results = new DefaultTableModel(Result.tableColumns, 0);
  Algorithm algorithm;
  Parameters parameters;
  ArrayList<Pair<JTextField, Holder<Double>>> parameterBindings = new ArrayList<>();

  private JFrame frame;
  private JTextField textDate;
  private JTextField textOrigin;
  private JTextField textAge;
  private JTable tableResults;
  private JProgressBar progressBar;

  /**
   * Launch the application.
   */
  public static void main(String[] args) {
    DatabaseManager.init();

    EventQueue.invokeLater(new Runnable() {
      @Override
      public void run() {
        try {
          Application window = new Application();
          window.frame.setTitle("Meeting point planner");
          window.frame.setVisible(true);
          window.textOrigin.requestFocus();
        } catch (Exception e) {
          e.printStackTrace();
        }
      }
    });
  }

  /**
   * Create the application.
   */
  public Application() {
    initialize();
  }

  /**
   * Initialize the contents of the frame.
   */
  private void initialize() {
    frame = new JFrame();
    frame.setBounds(100, 100, 940, 550);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    JPanel panelMain = new JPanel();
    frame.getContentPane().add(panelMain, BorderLayout.CENTER);
    panelMain.setLayout(null);

    JPanel panelInput = new JPanel();
    panelInput.setBounds(10, 11, 309, 489);
    panelInput.setBorder(BorderFactory.createTitledBorder("Input"));
    panelMain.add(panelInput);
    panelInput.setLayout(null);

    JLabel lblDate = new JLabel("Date (YYYYMMDD):");
    lblDate.setBounds(10, 21, 94, 14);
    panelInput.add(lblDate);

    textDate = new JTextField();
    textDate.setText("20170815");
    textDate.setBounds(114, 18, 78, 20);
    panelInput.add(textDate);
    textDate.setColumns(10);

    JLabel lblAttendants = new JLabel("Attendants:");
    lblAttendants.setBounds(10, 46, 78, 14);
    panelInput.add(lblAttendants);

    JList<Attendant> list = new JList<Attendant>();
    list.setModel(attendants);
    list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

    JScrollPane scrollAttendants = new JScrollPane(list);
    scrollAttendants.setBounds(10, 71, 289, 249);
    panelInput.add(scrollAttendants);

    JLabel lblOrigin = new JLabel("Origin:");
    lblOrigin.setBounds(10, 331, 46, 14);
    panelInput.add(lblOrigin);

    textOrigin = new JTextField();
    textOrigin.setBounds(66, 328, 114, 20);
    panelInput.add(textOrigin);
    textOrigin.setColumns(10);

    JLabel lblAge = new JLabel("Age:");
```

```
    lblAge.setBounds(10, 356, 46, 14);
    panelInput.add(lblAge);

    textAge = new JTextField();
    textAge.setBounds(66, 353, 114, 20);
    textAge.addActionListener(this::addAttendant);
    panelInput.add(textAge);
    textAge.setColumns(10);

    JButton btnAdd = new JButton("Add");
    btnAdd.addActionListener(this::addAttendant);
    btnAdd.setBounds(10, 381, 78, 23);
    panelInput.add(btnAdd);

    JButton btnRemove = new JButton("Remove");
    btnRemove.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
        int selectedIndex = list.getSelectedIndex();
        if (selectedIndex != -1) {
          attendants.remove(selectedIndex);
        }
      }
    });
    btnRemove.setBounds(94, 381, 86, 23);
    panelInput.add(btnRemove);

    algorithm = new DiscreteAlgorithm(10, d -> progressBar.setValue((int) (d * 1000)));

    JPanel panelResult = new JPanel();
    panelResult.setBounds(544, 11, 370, 489);
    panelResult.setBorder(BorderFactory.createTitledBorder("Result"));
    panelMain.add(panelResult);
    panelResult.setLayout(null);

    JButton btnCalculate = new JButton("Compute");
    btnCalculate.setBounds(20, 24, 89, 23);
    panelResult.add(btnCalculate);
    btnCalculate.addActionListener(e -> executor.submit(this::compute));

    progressBar = new JProgressBar();
    progressBar.setBounds(119, 28, 226, 14);
    panelResult.add(progressBar);
    progressBar.setMinimum(0);
    progressBar.setMaximum(1000);

    tableResults = new JTable();
    tableResults.setModel(results);

    JScrollPane scrollResults = new JScrollPane(tableResults);
    scrollResults.setBounds(20, 59, 325, 353);
    panelResult.add(scrollResults);

    JPanel panelParams = new JPanel();
    panelParams.setBounds(329, 11, 205, 489);
    panelParams.setBorder(BorderFactory.createTitledBorder("Parameters"));
    panelMain.add(panelParams);
    panelParams.setLayout(null);

    parameters = new Parameters();

    int positionY = 18;
    int additionY = 26;
    for (Map.Entry<Class<? extends ProductivityMap>, ParametersList> entry : parameters
        .entrySet()) {
      String name = entry.getKey().getSimpleName();
      name = name.substring(0, name.length() - 3);
      ParametersList parameters = entry.getValue();

      final JCheckBox chckbx = new JCheckBox(name);
      chckbx.setBounds(6, positionY, 150, 23);
      positionY += additionY;
      chckbx.setSelected(parameters.getUseThisMap());
      chckbx.addActionListener(e -> parameters.setUseThisMap(chckbx.isSelected()));
      panelParams.add(chckbx);

      for (Map.Entry<String, Holder<Double>> parameter : parameters.entrySet()) {
        final JLabel label = new JLabel(parameter.getKey());
        label.setBounds(32, positionY, 59, 14);
        panelParams.add(label);

        final JTextField textField = new JTextField();
        textField.setBounds(100, positionY - 3, 86, 20);
        textField.setColumns(10);
        textField.setText(parameter.getValue().get().toString());
        parameterBindings
            .add(new ImmutablePair<JTextField, Holder<Double>>(textField, parameter.getValue()));
        panelParams.add(textField);

        positionY += additionY;
      }
    }

  }

  private void addAttendant(ActionEvent e) {
    try {
      Place origin = PlaceFinder.city(textOrigin.getText());
      double age = Double.parseDouble(textAge.getText());
      Attendant at = new Attendant(origin, age);
```

```
245       attendants.addElement(at);
246       textOrigin.setText("");
247       textAge.setText("");
248     } catch (NotFoundException e1) {
249       // do nothing
250     } catch (SQLException e1) {
251       e1.printStackTrace();
252     }
253     textOrigin.requestFocus();
254   }
255
256   private void compute() {
257     if (attendants.size() > 0) {
258       try {
259
260         for (Pair<JTextField, Holder<Double>> binding : parameterBindings) {
261           try {
262             Double value = Double.parseDouble(binding.getLeft().getText());
263             binding.getRight().set(value);
264           } catch (NumberFormatException e) {
265             e.printStackTrace();
266             return;
267           }
268         }
269
270         ProductivityMapsFactory mapsFactory = new ProductivityMapsFactory(parameters);
271         for (Map.Entry<Class<? extends ProductivityMap>, ParametersList> entry : parameters
272             .entrySet()) {
273           ParametersList list = entry.getValue();
274           if (list.getUseThisMap()) {
275             Class<? extends ProductivityMap> clazz = entry.getKey();
276             mapsFactory.addFactory(clazz);
277           }
278         }
279
280         try {
281           SimpleDateFormat parser = new SimpleDateFormat("yyyyMMdd");
282           Date date = parser.parse(textDate.getText());
283           ArrayList<Attendant> ats = Collections.list(attendants.elements());
284
285           TreeSet<Result> resultSet =
286               algorithm.find(date, ats.toArray(new Attendant[0]), mapsFactory);
287           Object[][] rows = new Object[resultSet.size()][];
288           int i = 0;
289           for (Result r : resultSet) {
290             rows[i] = r.getTableRow();
291             i++;
292           }
293           results.setDataVector(rows, Result.tableColumns);
294         } catch (ParseException e1) {
295           e1.printStackTrace();
296         }
297       } catch (Exception e) {
298         e.printStackTrace();
299       }
300     } else {
301       System.out.println("No attendants added");
302     }
303   }
304 }
```

src/main/java/com/jjurm/projects/mpp/util/Holder.java

```
1  package com.jjurm.projects.mpp.util;
2
3  public class Holder<T> {
4
5    protected T value;
6
7    public Holder(T value) {
8      this.value = value;
9    }
10
11   public T get() {
12     return value;
13   }
14
15   public void set(T value) {
16     this.value = value;
17   }
18
19 }
```