SuperScript

04 January 2022

Version 1.2

Simulation engine for a social teamwork game

This document provides in-depth details of the SuperScript agent-based model of team formation. The version described here is the current stable release (version 1.2.). A more succinct introduction to the model and how to use it can be found in the README file at:
https://github.com/Superscriptus/SuperScript

Contents

1. Motivation

1.1 Model purpose

The purpose of the extended model is to build a simulation engine that serves as the basis for the “social teamwork game”. The model simulates the state of the world and its development over time for workers within an organization that are assembled in teams to work on projects.

The simulation engine will then serve as the underlying model for the world in a Python-Django prototype for a web application (e.g., hosted on Heroku). In that prototype a user (either a worker or an organization) goes through the steps of the user journey and takes part in the social teamwork game.

The model will be used to test a number of hypotheses to understand emergent properties and/or to fine-tune the input parameters, to visualize those results, to generate training data for the Machine Learning prototype and to gather model proof points for further discussion.

1.2 Baseline model

The baseline model is the NetLogo team assembly model developed by Wilensky, based on research by Guimera et al. The baseline model represents a business network of co-worker agents from which project teams of various sizes form. Agents in the model have only a few basic characteristics that influence their behavior: whether they are a newcomer or incumbent and what previous connections they have with other agents if they are incumbents. There are three parameters that can be adjusted to influence behavior in the baseline assembly model: the team size, the probability of choosing an incumbent, P, and the probability of choosing a previous collaborator, Q.

2. Model overview

2.1 Introduction

For this model, we consider the multi-project environment of a typical organization with multi-skilled workers who differ in their skill levels. The task of project staffing is to compose project teams such that the skills and availabilities of the worker meet the requirements of the respective project. A greater skill level does not reduce the time needed to accomplish a certain amount of workload but increases outcome quality, and in turn, the probability of project success.

In line with the literature, we argue that projects should be accomplished by relatively small teams and that workers should be assigned to a preferably small number of project teams.

A project is defined by its skill requirements (units of skill-by-level), a required creativity level, the project risk, and timing and budgetary constraints. For each project, skill requirements arise in the periods of its execution. Workload does not only originate from projects but also arises within the departments of the organization. Departmental workload must be accomplished in each period of the planning horizon by the workers who belong to the corresponding department. Each worker belongs to exactly one department.

Hence, we presume a matrix organization that features functional departments and, potentially, cross-departmental project teams. Our goal is to find an assignment of workers to projects to allocate project workload such that all requirements of projects and departments are satisfied and that the average probability of project success is maximized. The probability of project success is a function of the average rating of the required skills, a skill balance, a creativity match, and a chemistry booster.

We consider an organization that intends to carry out a set of projects within the upcoming planning horizon. The link between the organization and the projects are skills that are mastered by the workers of the organization and that are required by the projects. The organization wants to allocate project workload to its workers such that the average probability of project success is maximized.

2.2 Implementation

The model is implemented in Python 3.6 using the Mesa agent-based modelling framework. The allocation of near-optimal teams of workers to projects involves a complex numerical optimization task. For this task we use the basin-hopping algorithm from SciPy.optimize with Pathos multiprocessing to speed up the optimization by leveraging multi-core architectures. Heuristically we have found that this approach produces significant runtime improvements when using 8 cores/processes, with diminishing returns for larger numbers of cores.

2.3 Main concepts

2.3.1 Workers

Workers are the agents of the model. They can: work on projects; contribute to departmental workload; train their skills and be replaced by new workers if they are inactive for too long. The SuperScript worker class is derived from the Mesa.agent class.

2.3.2 Departments

Each worker belongs to a department. In general, the organization consists of D departments and there is a baseline departmental workload that is required in order to keep the department running. This workload must be met by the department's workers and acts as a constraint on the capacity of the workers to contribute to projects.

2.3.3 Projects

A certain number of projects are created on each timestep of the simulation. Project creation is either done at random, or by loading pre-defined projects (for repeatability). Each project has a specific combination of skill requirements, along with the following attributes that characterize the project: *risk, required creativity, budget.*

2.3.4 Team Allocator

A TeamAllocator class assembles workers into teams to work on projects by trying to find the best team of workers according to the project requirements. The team allocation uses predefined strategies (e.g. *Random*, *Basic, Basin*) which can be selected by the user. The general aim in team allocation is to maximize the probability of project success. However, this is a computationally hard problem because of the combinatorically large number of possible teams (there are 8.25×10^{12} teams of 5 that can be

selected from 1000 workers). Therefore, the various strategies that have been implemented attempt to simplify this problem in different ways.

2.3.5 Probability of success

The probability of project success is the key objective function that is optimized during team allocation. The probability is a function of the average rating of the required skills, a skill balance, a creativity match, and a chemistry booster. These components are defined in section 3.4.

2.3.6 Trainer

Low-skilled workers are trained to improve their skill levels. Training is blocking meaning that workers cannot work on projects or contributes to departmental workload while they are on training. The training process is handled by the Trainer class, which can operate on two different modes. The default mode aims to keep a fixed fraction of the workforce in training at any given time.

2.4 Main dynamics

2.4.1 Initialization

When the model is constructed the following initialization steps are taken:

- The main model parameters are loaded from the config file.
- A project inventory is instantiated, which handles project creation and tracks project status. It also contains the team allocator which is used to allocate teams on project creation.
- The workforce is created by generating the required number of workers and randomly assigning their skill levels. On creation, each worker is assigned at random to a department such that there are equal numbers of workers belonging to each department.
- The social network is instantiated, which tracks the number of successful collaborations between each pair of workers.
- The Mesa scheduler is instantiated, which is responsible for updating all of the agents (workers) on each timestep by calling their *step()* method. We use the *RandomActivation* schedule such that the workers are updated in a random order on each timestep.
- The trainer is instantiated, which handles all training of workers throughout the simulation.
- The data collector (SSDataCollector) is instantiated, which tracks various model-, agent- and project-level variables throughout the course of the simulation. These tracked variables are used by the Mesa server for visualization of the simulation in real time, or can be used to access data at the end of the simulation for plotting of saving to disk.

2.4.2 Running

Once the model has been initialized it can be run for a set number of timesteps by calling *run_model(step_count).* Alternatively the state of the simulation can be advanced by a single timestep by calling the model's *step()* method. Each time the *step()* method is called, the following events take place in this order:

- All workers' skill change trackers are reset (see section 3.1)
- The skill quartiles, used by the trainer, are updated (see section 3.6).
- The inventory creates new projects, and a team of workers is assigned to each (see section 3.5). Projects for which a valid team cannot be found are labelled as null projects (see section 3.3).
- The probability of project success is determined for each project that has a valid team (see section 3.4).
- The *schedule.step()* method is called, advancing each worker's state (see section 3.1). Workers that have been assigned as team leaders advance the state of the project(s) that they lead. If a project reaches completion, its *terminate()* method is called: the project outcome is determined and recorded; worker skills are updated; and the project is removed from the inventory.
- Workers enter training according to the training mechanism that is in use (see section 3.6).
- Null projects are removed from the inventory.
- The social network is saved to disk (if this functionality is switched on in the config file – the frequency at which to save the network can also be controlled).
- The data collector records the tracked variables (see section 4.2).

3. Details and definitions

In this section we provide definitions for the various model elements that were introduced above. Throughout this section we use the default values of the model parameters as defined in the config file of model version 1.2. Were concepts are introduced, we also provide links to the relevant python files that contain the corresponding class or method.

3.1 Worker attributes and methods

Each worker has the following key properties:

3.1.1 Skills

The worker gets a set ("endowment") of initial skills *s* at different levels *l(ks)*. Skills A-E are so-called hard skills and are required directly by projects. Skills F-J are soft skills and are used to determine the cognitive diversity of a team. The initial skill set is determined as follows:

1. For hard skills (A-E), it is determined uniformly at random whether the skill is existent (1) or not (0). The probability of skill existence is 80%, that is on average workers possess 4 out of 5 hard skills. However, it is ensured that each worker has at least one existent skill.
2. For existing hard skills, a level is assigned as a random number between 1 and 5. This is rounded to one decimal place for display purposes.
3. For soft skills (F-J), a level is assigned as a random number between 1 and 5, again rounded to one decimal place for display.

Example of skill matrix of a worker *k*:

Skill	**A**	**B**	**C**	**D**	**E**	F	G	H	I	J
Level		**3.9**	**3.2**	**4.1**	**1.5**	4.9	3.0	4.1	1.2	3.5
None	**0**									

Soft skills are static and cannot change during a simulation. Hard skills are updated based on project work experience (via *peer assessment* – see section **Error! Reference source not found.**), training and s kill decay. Skill decay is where unused skills are subject to a multiplicative decay of 0.99 on each timestep.

3.1.2 OVR

The worker overall rating (OVR) is the average of the hard skills, multiplied by 20. In this example, the worker OVR is 63.5. This is used as a measure for how highly skilled the worker is.

3.1.3 Strategy

The worker strategy is used by the worker to bid on projects. The default strategy is called ‘stake’ and means that workers can only bid for projects where the risk is less than or equal to half of their OVR.

3.1.4 Project list

This is a list (called ‘*leads_on’*) of the projects which the worker is currently leading on. Project leaders are responsible for advancing the state of the projects that they lead on.

3.1.5 Contributions

This class uses a dictionary to track the contributions that the worker makes to different projects throughout the course of the simulation. These contributions are determined during team allocation. The worker can only work up to a maximum of 100% full time equivalent (FTE). Each skill that they contribute to a project accounts for a single *unit* which is equal to 10% FTE.

3.1.6 History

This class tracks the workers recent success rate on projects. The success rate is used to compute a quantity called *momentum,* which is used in the probability calculation (see section 3.4).

3.1.7 Training remaining

This integer counter indicates the number of timesteps for which the worker will be on training. If the worker is not currently on training, then the counter is equal to -1. A single training course lasts for 5 timesteps (see section 3.6).

3.1.8 Timesteps inactive

This integer counter records the number of timesteps for which the worker has not contributed to any projects. If the worker is inactive for 10 timesteps, they are replaced by a new worker.

3.1.9 Skill change trackers

These dictionaries log the change in each hard skill that is produced by each of the three mechanisms (peer assessment, skill decay, training) on a single timestep. The tracker are reset are the beginning of each timestep.

3.2 Departments

Departmental workload requirement is expressed as a percentage of total department capacity and is uniform across departments. We presume that the work requirement of a department can be accomplished by an arbitrary subset of its workers and that every worker of department performs departmental work with the same efficiency. So, a workload of 10% could be met by all of the department's worker reserving 10% of their time for departmental work. Alternatively, a department of 100 workers could meet this requirement by allocating all departmental workload to 10 of its workers, for example.

3.3 Project attributes and methods

3.3.1 Project creation

The planning horizon of the firm is defined as the largest number of possible timesteps from project creation to project completion. Under default parameters, the length of the planning horizon is 10 timesteps: the length of each project is 5 timesteps, and there are 5 possible start times for the project (including the current timestep).

Projects are created at random on each timestep by the project inventory. On project creation, its start time is determined; its requirements are determined, and a team is allocated to the project. If team allocation is successful, the probability of project success is calculated.

3.3.2 Project requirements

A project is defined by the following five factors: (1) its minimum requirements for hard skills, (2) the required creativity level, (3) the project risk, (4) budgetary constraints and (5) timeline constraints.

(1) Required hard skills A-E, are defined as units (one unit equals 0.1 full-time equivalents) of skills-by-level. For the sake of simplicity, both required skill level and number of units are integers. The required hard skills can be represented as a matrix, which is determined as follows:

   (i) It is determined whether each skill is required (1) or not (0). The probability of a skill requirement is 80%. That is, on average, a project requires 4 out of 5 hard skills.
   (ii) Skill units are allocated randomly into the 5 levels, whereby no skill can require more than 7 units and per skill only one level is allocated. Also, for at least one skill more than 2 units are required.
   (iii) The maximum number of skill units a particular worker can contribute is limited to 10 across all skills (i.e., a skill unit is equivalent to 0.1 full-time equivalents). So, in the example below, two skill units at level 4 for skill B are required by the project. To meet this requirement would require two workers with B skill levels of ≥ 4 to contribute 10% (i.e. a single unit) of their time each.

Skill / level	**A**	**B**	**C**	**D**	**E**
5					
4		**2**		**4**	
3			**3**		**2**
2					
1					
Not required	**0**				

(2) The required creativity level is defined by a random whole number between 1 and 5.

(3) Project risk is defined by the required staking and randomly determined as either 5, 10 or 25.

(4) Budgetary constraints are expressed as the multiplication of hard skill levels and units required of those skills, e.g., in the example above the project budget is 39 (= 2*4 + 3*3 + 4*4 + 2*3). While for 75% of projects this budget is the upper limit, in 25% of cases there is budgetary flexibility which allows the budget to increase by 25% (e.g., the budget increases from 39 to 49).

(5) The project start time can be limited to immediate (time=0) or be extended by 1, 2, 3 or 4 timesteps. Half of the projects have no timeline flexibility and the remainder have a potential for delayed start time. The intended probabilities for potential project start time offsets are shown in the following table:

Time step	0	1	2	3	4
Frequency	50%	25%	10%	10%	5%

In the function definitions notebook, we illustrate the decay function that is used to approximate these start time offsets.

Timing flexibility is valuable as it allows the team assembly process not just to focus on t=0 worker availability but extend it to future periods with the possibility of being able to assemble a team with a higher success probability (“forward dispatch”). Workers that are pre-booked for future timesteps are not available for training or other projects.

3.3.3 Project termination

The project ends and is settled. At this point, it is determined whether the project was successful or not. The stake is returned, and the utilized hard skills are updated based on project success and peer reviews. (Note that skill level values below are given to only one decimal place for clarity.)

The settlement procedure consists of the following steps:

1. Success of the project (yes/no) is determined by selecting a uniform random number on the interval [0,1]. If the selected number is ≤ to the expected project success probability (see section 3.4) then the project is successful.

2. A peer-to-peer skill assessment takes place, which models the team mates as well as the organization rating the skill level of the worker. The average of these assessments is combined with the existing skill level with a 25% weight. The deviation of these assessments from the original skill level depends on the project success. The assessment factor is a normally distributed random number with the following [mean, stdev] properties (these distributions are illustrated in the function definitions notebook):

project success	yes	no
assessment factor	1.05, 0.2	0.95, 0.2

   E.g., For a successful project, the utilized skill D of a worker was originally at level 4.1. This level is updated on average by factor 1.05 and a weighting factor of 25%. Hence, the average skill level for skill D after the peer-to-peer assessment is 4.2 = 0.75 * 4.1 + 0.25 * 1.05 * 4.1

3. This updated level-by-skill is then multiplied by a factor that depends on project success and project risk (i.e., the required staking of 5, 10 or 25)

project success	no	Yes (5)	Yes (10)	Yes (25)
success factor	1.0	1.05	1.1	1.25

   E.g., For the worker whose level of skill D was updated from 4.1 to 4.2 after the peer assessment, it is then updated to 4.6 (= 4.2 * 1.1 for a successful medium risk project, i.e., “Yes (10)”).

4. The level-by-skill is capped at 5 for each worker.
5. The social graph is updated (three-dimensional vector between any two workers, indicating number of projects, success ratio of projects and average of reciprocal ratings).

3.4 Probability function

The objective function for the optimization that is conducted during team allocation is to:

- Maximize average expected probability of project success, ***P(success)***, at timestep *t*
- Subject to:
  - Time workers spend on projects does not exceed their availabilities.
  - Sufficient time is spared for departmental requirements.
  - Team size is between 3 and 7.
  - Time and budgetary constraints of the project are met.
- Where ***P(success)*** is a function of the average rating of the required skills, a skill balance, a creativity match and a chemistry booster.

The probability of project success, ***P(success)***, is an additive function of five elements: (1) the team OVR, (2) the skill balance, (3) the creativity match, (4) a chemistry booster, and (5) the project risk. The probability is defined as:

$$P(success) = \sum_{i=1}^{5} f_i(\text{element}_i),$$

where $f_i(\text{element}_i)$ is a function of the *i*th element that determines its contribution to the total probability. We define these five elements and their corresponding functions below.

Once the raw probability value ***P(success)*** has been calculated it is then normalized using max-min scaling such that the value lies in the closed interval [0, 1]. This normalization uses the formula: $P_{norm}(success) = (P - P_{min}) / (P_{max} - P_{min})$, where P_{max} and P_{min} are respectively the maximum (0.95) and minimum (-1.075) theoretical values for ***P(success)*** based on the components defined below. The normalization procedure is demonstrated in the probability components notebook.

3.4.1 Team OVR

The team overall rating (OVR) is the average skill level for the required skills multiplied by 20. E.g., in the example below, the project requires 11 skill units (i.e., 1.1 full-time equivalents) and those are allocated as follows (purple shade) in the table below. Therefore, the team OVR is 72.36.

Skill	**A**	**B**	**C**	**D**	**E**
Worker 1		**3.9**	**3.2**	**4.1**	**1.5**
Worker 2	**4.1**	**4.4**	**2.1**	**2.9**	**0.4**
Worker 3				**0.5**	**3.9**
Worker 4		**3.6**	**2.5**	**4.9**	**5.0**
Worker 5				**2.9**	
Total units	**0**	**2**	**3**	**4**	**2**

The team OVR contributes to the probability score according to the function:

$$f_i(teamOVR) = \frac{0.75 \times teamOVR}{100},$$

such that a maximal team OVR of 100 contributes a success probability of 0.75.

3.4.2 Skill balance

The skill balance (aka ‘degree of required skills match’) is defined as the sum of squared *negative* differences between the team’s actual hard skills (average level by skill) and the required hard skills (required hard level by skill), divided by the number of required skills with negative differences. E.g., for skill C 3 units with level ≥ 3 are required while the actual average team skill is 2.6 (3.2 + 2.1 + 2.5 / 3).

Therefore, the squared negative difference is 0.16 (-0.4^2). The same procedure is applied to all skills and the average of negative squared differences is taken.

The skill balance takes values on [0,16] and contributes to the probability score according to the function:

$$f_i(skillBalance) = \frac{-2.5 \times skillBalance}{100},$$

such that a maximal skill balance of 16 produces a negative contribution of 0.40.

3.4.3 Creativity match

The degree of creativity match is defined as the squared difference between the team's actual creativity level and the required creativity level of the project. The creativity level of a team T is a measure of the team's heterogeneity in terms of soft skills and is defined as:

$$creativity(T) = \frac{1}{5}\sum_{s=1}^{5}\frac{1}{|P_s|}\left(\sum_{(i,j)\in P_S}\frac{(k_{s,i} - k_{s,j})}{D},\right)$$

where $k_{s,i}$ is the skill level of team member i for soft skill s; P_s is the set of pairs of team members with soft skill s; and $D = (\max(k_s) - \min(k_s)/(|T| - 1)$ is the maximum possible distance between skill levels in a team of $|T|$ workers (in the maximum entropy/heterogeneity configuration).

The above formula gives a creativity level on [0,1], which is then mapped on to [1,5] using the transformation:

$$creativity'(T) = 1 + (creativity(T) \times D)$$

The creativity match is then defined as the squared difference between this transformed creativity value and the creativity level require by the project:

$$creativity\ match(T, P) = (creativity'(T) - creativity(P))^2$$

This takes values on [0,16] and contributes to the probability score according to the function:

$$f_i(creativity\ match) = \frac{1 - 4\,(1 - e^{-creativity\ match})}{10},$$

which is a saturating function that tends asymptotically to -0.3 as depicted in the function definitions notebook.

3.4.4 Chemistry booster

The chemistry booster has four components: three for the individual worker level, and one for the team level. Each component is 1 if positive and 0 if negative. The components are summed to arrive at the chemistry level.

- Chemistry at individual worker level is impacted by:
    - Relevant position and role for worker: a position is relevant if at least one of the worker's two top skills, the primary and secondary skill, is required in the project.
    - High stake level: required staking is >= 10% of a worker's OVR
    - Momentum, e.g., 3 out of 4 last projects have been successfully completed.
- Chemistry at team level is impacted by:
    - Links between team members who worked on successful projects in the past is >= 50% (that is, half of the teammates have worked together before on successful projects).

The chemistry level, which takes values on [0,4] is then used to compute the probability contribution via the formula:

$$f_i(chemistry\ level) = \frac{2.5\ \times chemistry\ level}{100}$$

3.4.5 Project risk

Project risk is measured by the required staking and takes one of three values: 5, 10, 25.

The risk contributes to the probability via the formula:

$$f_i(risk) = \frac{15 - (3\ \times risk)}{400},$$

which is equal to -0.15 as depicted in the function definitions notebook.

3.4.6 Example

The five components of the probability score are summed. If the result is less than 0, the probability is set to zero since negative probabilities are not meaningful.

As an example, a team with a team OVR of 58, a degree of required skills match of 0.5, a degree of creativity match of 1.5, a chemistry level of 3 and a required staking of 5 will have an expected success rate of 30% (i.e., 45% + (-5%) + (-15%) + 5% + 0% = 30%).

3.5 Team allocation

The intention for team allocation is to find the project team with the highest predicted success rate utilizing the available pool of workers, subject to budgetary and time constraints. Also, the departmental workload must be met. This makes it a difficult constrained optimization with a large solution space. We have implemented several strategies that approximate solutions to this optimization problem and intend to improve on these solutions in the future.

Team allocation is a two-stage process. First workers are invited to bid for projects, and then a team is allocated by selecting workers from the resulting bid pool. The worker bidding behavior is determined by a *worker strategy*. The team allocation behavior is then determined by an *organization strategy*. Currently the best organization strategy is *'Basin'* in terms of maximizing the probability of project success. However, this strategy is slow and requires multiple CPU cores. For rapid simulations or real-time visualization in the Mesa frontend (see section 4.1), we currently recommend using either the *'Random'* or *'Basin'* organization strategies.

3.5.1 Worker strategies

Only workers that are not engaged in training and that have available skill units to offer are able to bid on projects. We have implemented two worker strategies: (1) *'AllIn'* and (2) *'Stake'*.

In the *'AllIn'* strategy, workers bid for all projects provided that they are available for project work. In the *'Stake'* strategy workers must have a sufficiently high OVR (overall rating) in order to bid. This is defined as the project risk is < 50% of the worker OVR.

3.5.2 Organization strategy: Random

The random strategy simply selects team members are random from the bid pool, producing a team of between 3 and 7 workers. This selection process often exceeds the project budget resulting in many *null projects.*

3.5.3 Organization strategy: Basic

The basic strategy sorts the bid pool in descending order of total skill level (summed across the skills required by the project). It then produces a team by selecting the top N workers from the sorted bid pool, up to a value of N that either exceeds the project budget or the maximum team size.

3.5.4 Organization strategy: Basin

The basin strategy uses SciPy's basinhopping algorithm with the COBYLA optimizer on each step, to find teams that maximize the project probability of success. The intention is that the COBYLA algorithm finds the local optimum and the basinhopping steps moves the solver to new regions of the solution space where better local optima may be found. The is no guarantee of finding the global optimum team allocation and the solutions generally improves with the amount computational effort. The basinhopping runs are conducted in parallel, on a number of cores defined by *NUMBER_OF_PROCESSORS.* Each run does a number of hops that is defined by *NUMBER_OF_HOPS.* The solver starts from an initial *'smart guess'*, which is produced by a weighted random selection from the bid pool where the weighting is assigned based on the distance between the worker's skill vector and the skills required by the project. The same weighted random selection is used on each basinhopping step to randomly shuffle the team by adding and removing workers.

3.6 Training mechanism

Workers are only eligible for training if they are not involved in project work; not booked for project work during the length of the training; and there is sufficient departmental capacity to conduct the departmental workload. The workers selected for training are those with the lowest skill level over the two top priority skills at that timestep. Priority skills are defined as those most in need for projects at that timestep in terms of the sum of required units of each skill. The are two training modes: (1) 'all' and (2) 'slots'.

3.6.1 Mode: all

In this training mode, all workers who are below the median skill level for either of the top two priority skills enter training.

3.6.2 Mode: slots

At every time step, a fixed number of "training slots" made available. Training is allocated to the workers whose distance to median for the priority skills is largest. The number of training slots made available at every time step is set such that in steady state 10% of the workforce is on training. E.g., if the workforce consists of 100 workers, the number of training slots is 2 = 10%*100/5.

3.6.3 Training mechanism

Only one skill *s* is trained per training course. The length of a training course is 5 timesteps. During this time, the worker is not available for project work nor for departmental work. The training will upgrade the

skill level to the first quartile level of all workers in the organization. Currently this upgrade occurs immediately on entering training.

4. Visualization and data output

4.1 Mesa server

The simulation may be run using the Mesa browser-based live visualization by invoking *mesa runserver*. In this case, various model parameters can be set using the GUI controls and the simulation can be stopped/started or stepped through step-by-step. The display and control elements are defined in the server script. Importantly, this is where the network visualization can be switched on or off. This is desirable because for large simulations (i.e. many workers) the network visualization slows the simulation down by having the re-compute the node layouts on each timestep.

4.2 DataCollector

The class SSDataCollector is a subclass of the Mesa data collector and is used to track model-, worker- and project- level variables throughout the simulation. These tracked variables can be accessed at any time using the Mesa.DataCollector *get* methods.

In the current version (v1.2) we have introduced a metric called “Return on Investment” (ROI), which is one measure of how well an organization is operating. A script is provided to compute this metric retrospectively using pre-simulated data but future release will incorporate the metric into the main model. To calculate the ROI at timestep t, each worker is allocated the following return values for each of their 10 workload units based on the activity of that unit:

- 5% for ongoing project work
- 10% for departmental work
- 5% for ongoing training
- 50% for a successfully completed project
- 10% for an unsuccessful completed project
- 0% for inactive

We take the mean value of these returns across each worker’s 10 workload units, to produce the worker ROI. For example, a worker who completes a successful project to which they contributed two workload units and is also contributing four workload units to other ongoing projects as well as one workload unit to departmental work would have a worker ROI of (2 x 50 + 4 x 5 + 1 x 10 + 3 x 0) / 10 = 13%. We then take the mean of the worker ROI values across the workforce to produce the total ROI for the organization.

4.3 Running as a script

The simulation can also be created and run from within a Python script without directly invoking Mesa. The provided simulation runner scripts can be used to run the simulation for a set number of timesteps. In this case, the projects are saved to disk at the end of the *run_model()* method (if that functionality is activated in the config file) and the data collection variables can be saved to disk by the script. As standard, output data is saved to the local directory /simulation_io. Only sample data is included in this directory in the git repository in order to keep the repository size down.

5. Hypotheses

The following hypotheses regarding the dynamics of the simulation were proposed prior to implementation of the model. They are investigated using simulated data in the testing hypotheses notebook. We found that the model did produce the hypothesised behaviour, although in some cases the effect sizes were smaller or more subtle than expected.

(a) High risk projects (high stake) attract talent (high OVR)
    (i) Show OVR (individual and team) on y-axis and risk on x-axis
(b) Cognitively diverse teams have higher success rate than randomly selected teams
    (i) Show project success rate on y-axis and team creativity level (absolute) on x-axis
(c) Superstars emerge, i.e., workers who get over-proportionally selected to work in high stake projects, and, on the other hand, workers with low OVR are stuck with departmental workload and, over time, get replaced by new workers
    (i) Show (scatterplot) OVR (individual) on y-axis and risk (stake) on x-axis
    (ii) Show (bars) OVR (individual) on y-axis and %-age of idle, departmental and project workload
(d) Timeline flexibility pays off (start date can be later) in terms of higher project success rates
    (i) Show (scatter) project value add on y-axis (with and without timing flexibility) and project budget on x-axis (DONE)
(e) Budgetary flexibility pays off (higher budget) in terms of higher project success rates
    (i) Show (scatter) project value add on y-axis (with and without budgetary flexibility) and risk on x-axis
(f) Targeted training pays off in terms of higher project success rate and lower turnover
    (i) Show (scatter) project value add on primary y-axis (with and without training), turnover on secondary y-axis, and time on x-axis
(g) Some slack in the system, i.e., a utilization rate of workers <100% will add value
    (i) Show (line) project value add on primary y-axis (with 10, 5, 2, 1 projects per timestep), average OVR on secondary y-axis, and time on x-axis
(h) Too much slack in the system will increase turn-over (i.e., too many workers are not engaged in projects or trainings, the median skill level drops and workers get replaced)
    (i) Show (line) turnover on primary y-axis (with 10, 5, 2, 1 projects per timestep), average OVR on secondary y-axis, and time on x-axis
(i) Low OVR workers stay low OVR and eventually get replaced, et vice versa (“the rich get richer” effect)
    (i) Show (line) fraction of workers with bottom-and first quartile OVR that get replaced on y-axis, and time on x-axis
(j) More and earlier training mitigates “the rich get richer” effect
    (i) See (i) but for training of all skills below median
(k) For high OVR workers, a high portion of their OVR change is due to project work experience while for low OVR workers a high portion of the OVR change is due to training
    (i) Show table with attribution of OVR change to the following factors for top and bottom quartile OVRs (measured at the end of the simulation period, e.g., after 100 time steps)
        (1) Starting value

(2) Decay

(3) Training

(4) Project work

(l) The higher the budgetary flexibility (up *and* down) and the lower the timing flexibility, the lower the latency (number of time steps a projects waits in the queue)

(i) Skipped

6. Extensions

(a) Allocation of required skill units to different skill levels for the same skill in the project requirements (e.g., to allow for junior and senior workers) [currently: skill units are only allocated to one level per skill]

(b) Remove the smart algorithm, i.e., not allowing for the required project skills to be smartly chosen by an ML powered algorithm. Doing so adds stochasticity around the project success probability, **prob**. Show that ML algorithm learns and over time increases **prob**. [currently: the smart algorithm is on and **prob** is a constant]

(c) Workers receive multiple invitations that exceed their availability and capacity (e.g., sum of assigned project units > 10 for a given moment in time *t*). That is, workers have to select from a list which project assignments to accept. Projects compete for workers and workers compete for projects. For this to work, a method for two-sided matching, a clearinghouse mechanism, is required, e.g., a Deferred Acceptance Mechanism. [currently: invitations do not exceed availability and capacity of a worker and hence all invitations get accepted]

(d) The project portfolio is determined endogenously, i.e., projects are selected (and prioritized) based on available skills in order to maximize their value add [currently: projects are determined exogenously]

(e) Long-lasting projects, i.e., more than 5 time steps, with changing skill requirements. The objective would be to minimize turnover within the team while meeting changing skill demands [currently: project length is set to 5 time steps with constant skill requirements]

(f) Training of a skill can be randomized, i.e., a random skill of the 5 skills is trained [currently: priority skills are trained when a particular worker's level is below medium]

(g) Priority skills can be set by the organization in order to support a bottom-up transformation [currently: priority skills are those two skills where the aggregate unit demand is highest]

(h) Introduce slack to avoid "overcommitment", e.g., sum of departmental workload, project workload and training engagements is set to never reach 90% of full utilization [currently: slack is not explicitly introduced]

(i) Viral mechanisms, e.g., feed and community for workers to see how their co-workers' journey (training, project assignments, OVR changes), to share best practices, to make rankings/top movers visible, to issue badges and to establish mentor and tribe relationships [currently: not implemented]

(j) Introduce a workforce engagement metric that increases project success probability, e.g., something like a virtuous circle [currently: not implemented]

(k) At the project settlement, the team is assessed collectively and not individually [currently: an individual peer-to-peer assessment is conducted]

(l) Method to assemble teams based on cost (“cheapest”) or average (“random”) [currently: team selection is aiming at maximizing project success probability]
(m) Model members of a team that randomly drop out during the project and have to get replaced (“hot swapping”) [currently: assembled team at initiation of the project stays together until the project ends]
(n) Model members of an active project team that are moved to another (higher priority) project that starts later (again, some type of “hot swapping”). This could result in a more “optimal path” [currently: assembled team at initiation of the project stays together until the project ends]
(o) Add stochasticity to r(dt) departmental workload requirement [currently: departmental workload requirement is deterministic]
(p) Project success is not binary but determined on a scale of -2, -1, 0, +1 and +2 [currently: project success is binary]
(q) Introduce objective (or constraint) to level the utilization across workers in the organization and / or within a department [currently: utilization can have corner solutions, e.g., “superstars” work 100% on projects while workers with low OVR conduct almost exclusively departmental workload]
(r) Introduce option for external hiring of workers with “defined” hard skills to close aggregate skill gaps [currently: model is limited to workers within the organizations and replacements are generated with random skill endowment]
(s) Budgetary flexibility is introduced to allow for lower team OVR (and thus, lower project success probability) [currently: budgetary flexibility increases the budget]
(t) Projects that are not staffed during the 5 time step planning period will be moved to the next planning period, and will create additional backlog projects [currently: projects that are not staffed during the planning period “get cancelled”]
(u) Introduce a constraint for maximum number of concurrent project participation for any worker, e.g., 2-3 as suggested by academic literature [currently: number of concurrent projects a worker is engaged in is not limited explicitly - there is however an implicit limit for high OVR workers via the staking requirement]
(v) Simulation of “shocks” to the system, i.e., introduction of a large high priority-high stake (“all hands on deck”) project (modelled, e.g., as the introduction of 10 new projects all with the same project requirements, and putting all workers on status “idle”) and analyse how those shock waves have knock-on effects on the remaining 50 projects in order to understand the fragility of the organization [currently: no exogenous shocks are modelled]
(w) Introduce possibility of changing soft skills [currently: soft skills vector remains constant over time]
(x) Introduce option to work on projects / or do departmental work while on training [currently: no project or departmental work is possible while on training]
(y) Make workers within a department more homogeneous in terms of skills (not level, but exposure to skill). Then, as a base-line (“benchmark”) randomly allocate projects of a department to workers of that department [currently: projects are not assigned to departments and workers are randomly endowed with skills]
(z) Introduce constraints in the form of one or multiple workers that are pre-assigned to the project team, i.e., the optimizer finds the most optimal team combination considering those constraints [currently: all workers are selected by the algorithm]
(aa) Introduce individual skill development options, i.e., ability of a worker to select skills they want to improve in and hence take trainings and/or get exposed to these skills in projects with team members that are more experienced [currently: not implemented]

(bb) Allow training triggers to change, i.e., instead of training being triggered by being below median for the top priority skills, it could be below first or fourth quartile [currently: trigger is set to median]