Simulation-based optimization of geometry and motion of a vertical tubular bag machine

Matthias Frank, Johann Holzweißig
matthias.frank@tu-dresden.de, johann.holzweissig@tu-dresden.de
Faculty of Mechanical Science and Engineering
Institute of Processing Machines and Mobile Machinery
Chair of Processing Machines and Processing Technology
Technical University of Dresden
Germany

Abstract

For food industry processes packaging machines with high throughput are required and one way to improve the overall machine efficiency is to increase its working speed. However, testing of prototypes is time and cost expensive. Therefore, simulation is used to evaluate the process and adapt it. Optimization can help to find better machine designs by using simulations to evaluate one solution.
This work uses the Discrete Element Method to model a vertical tubular bag machine for packaging basmati rice. The Covariance Matrix Adaption Evolution Strategy optimizes the simulation model and results in a significant machine speedup. This work is a guidance to adapt this method for similar problems.

1 Introduction

Granular materials such as beans, coffee or rice are widespread in our daily live. Packaging these material is often done with vertical tubular bag machines. Machine manufacturers strive to design faster and more reliable machines due to competitive pressure. De-

sign improvements are evaluated with experiments in the real world which are often time-consuming and expensive in the cost of materials. Simulation can help to reduce test material and the number of prototypes because only good solutions are build. Simulation allows also a look inside the model and visualize the results.

Optimization is an incremental process to improve prototypes or simulation models. A common way is to evaluate results of experiments and adapt it to get a better solution. Often this is done by an experienced engineer. Also, this process can be very time consuming. By evaluating the experiment's results automatically, a mathematical optimizer generates candidate solutions, evaluates them by simulation, compares the results and creates new candidate solutions. This process can be fully automated, so no or less user interaction is necessary.

The approach of simulation-based optimization with meta heuristics is here applied to granular food packaging. This article provides detailed guidance to optimize granular processes. It is indented to be easily reproducible even for new users in the field. To this end, we provide details about modeling, simulation, and optimization, provide hints, and point out common pitfalls.

In section 1 we describe the problem and our model. Section 2 gives an overview about particle simulation and its model calibration. Section 3 describes the theory behind our optimization technique, our implementation and the results.

1.1 Problem Description

A vertical tubular bag machine is used for dosing and packaging granular materials e.g. food like rice, sugar or coffee. The granulate is dosed using a cup with a defined volume. The cup moves below a storage which buffers

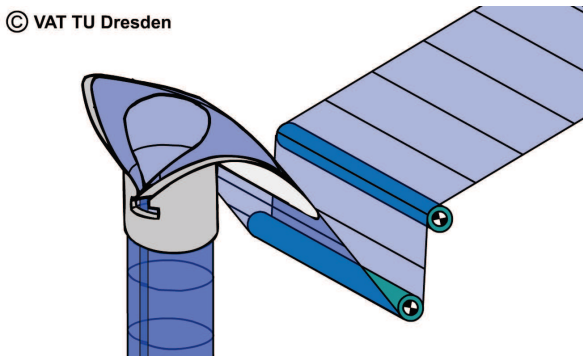

Figure 1: Schematic illustration of forming a plastic tube with a forming shoulder. [2]

the granular material. Then the cup is filled and is moved over the dropping hole and the granular material falls through a hopper and a tube into a plastic bag.

The plastic bag is made by pulling a plastic film over a forming shoulder to wrap it around the tube (figure 1). By vertically sealing the tube is closed.

Between each portion of granular material the tube is horizontally sealed and cut to get separate bags with product inside.

In this work, the granular material is Golden Sun Basmati rice and one portion is $1279 ml$. The aimed machine speed is $100 \frac{cycles}{minute}$ which corresponds to $600 ms$ per cycle. For sealing, including a safety margin, a time slot of $150 ms$ is necessary. So the rice must flow out at the end of the tube within $450 ms$. By scaling up the machine speed only by increasing the cycles, there is a high risk to violate the time slot for sealing. So it could happen that a few particles get into the sealed seam. This will lead to an incompletely closed bag which promotes pest infestation or spoil the product.

The aim of this work is to adjust the motion of the dosing cup and the hopper geometry to reach the desired machine speed by decreasing the filling time while keeping the safety

time slot.

When shorter machine cycles are used, dynamic effects like trajectory of granulate must be taken into account. So the hopper has to adjusted in such a way to allow the granular material get thrown through the hopper with least obstruction as possible.

To solve this problem a DEM simulation provides a process model for filling rice. The geometry is designed as a parametric CAD model. Exporting meshes and loading them into the simulator works automatically. A meta-heuristic optimizer iteratively generates new parameter sets for motion and geometry, the simulation evaluates these.

1.2 Geometry

The geometry can be designed with an ordinary parametric CAD application. It can be designed as a surface or a sheet[1]. For a fully automated optimization, an application with a macro API is necessary for exporting meshes without user interaction.

The choice is FreeCAD[2], an ambitious parametric CAD application in an early stage of development which bases on Open CASCADE[3]. Basic design features of such typical applications are implemented but some are missing until now like drawing or assemblies.

The main point of using FreeCAD in this work is its Python API. It can be used directly as a Python module and, therefore, it is possible to build or change parts directly from Python scripts. In this way changing data constraints in a FreeCAD part and the export of meshes can be implemented for fully automated operation.

The dropping hole construction is designed by defining some basic references. The hop-

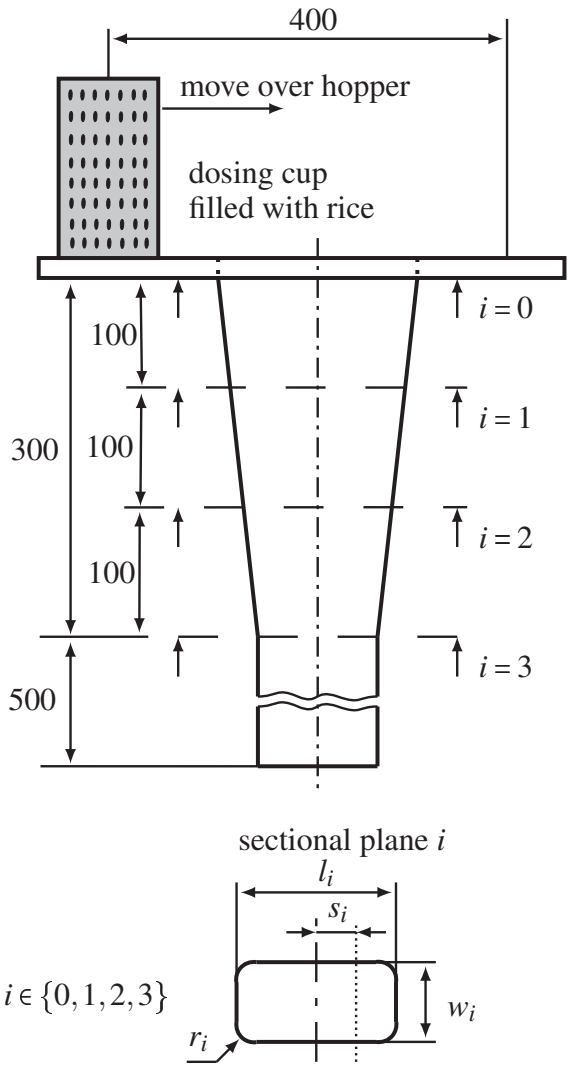

Figure 2: Schematic illustration of the geometric degrees of freedom. Measures in millimeters.

[1] Only the inner surface is described here.

[2] `http://www.freecadweb.org`, version 0.14

[3] `http://www.opencascade.org`

per, tube, cup and dropping hole are horizontally referenced by a dashed/dotted middle line. Each is described by a sectional plane i, illustrated in figure 2. Every sectional plane is a rectangle with corners rounded by radius r_i. The length l_i and width w_i defining the size of the rectangle. The shift dimension s_i positions the sectional plane regarding to the middle line. The dosing cup does not use a shift dimension. The tube at the bottom of the sketch has a fixed $s_3 = 0$ because shifting the tube will break connecting dimensions. Consequently it will not fit in the construction. The dropping hole is cut out of the base plane by sectional plane s_0 which is also the upper edge of the hopper. This geometric model enables the user to get a large variety of different geometries. With $s_i = 0$ and $l_{0,1,2} = l_3$, $w_{0,1,2} = w_3$ and $r_{0,1,2} = r_3$, the hopper adopts the same shape like the tube.

The hopper is conceived as a sheet metal part. Therefore only the inner surface is described. The FreeCAD feature Loft spans a surface over the sectional planes $s_{0,1,2,3}$. In addition, the manufacturability of the hopper has to be considered. Often the optimizer chooses values which lead to geometries which are hard or expensive to produce. Adding constraints is difficult and could handicap the optimizer by adding spatial limitations in the search space or inconvenient punishing terms. In the treatment described here, no limiting contraints of this type were used. If the optimizer does not result in an acceptable geometry, manual modification mitigates the issue.

1.3 Motion

In this work we use a simplified abstract model of a form-, fill- and sealing machine for simplicity to study the filling process. But the methodology is applicable to common machines which normally use circular dosing cup motion.

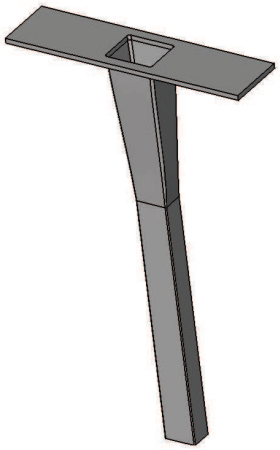

Figure 3: Initial Model of the Hopper

The dosing cup moves $400\,mm$ linearly over the hopper. Assuming a continuous working machine the easiest way is to move the cup with a constant velocity v. This would simplify the optimization problem by reducing degrees of freedom. In that case the optimizer will adjust the hopper shape to fit the cup motion. Assuming the usage of servo drives the adaption of motion is significantly easier to change than the geometry of machine parts. The main point of using non-linear motion is the compact dropping of the rice. For the desired high machine cycle frequency the rice has to fall compact through the hopper and tube into the bag. When the cup arrives the dropping hole and the intersection of cup and hole are small only some few grains fall through the hopper. During the cycle the intersection gets larger and more particles are dropped. At this point it could happen that too much rice falls into the hopper so the risk of bridges increases. The used Golden Sun Basmati rice forms no permanent bridges within this size of hoppers. But temporal bridges slow down the particle's velocity. If this happens the velocity of the dosing cup must be reduced to drop fewer rice grains. It would be beneficial to have at the beginning of the cy-

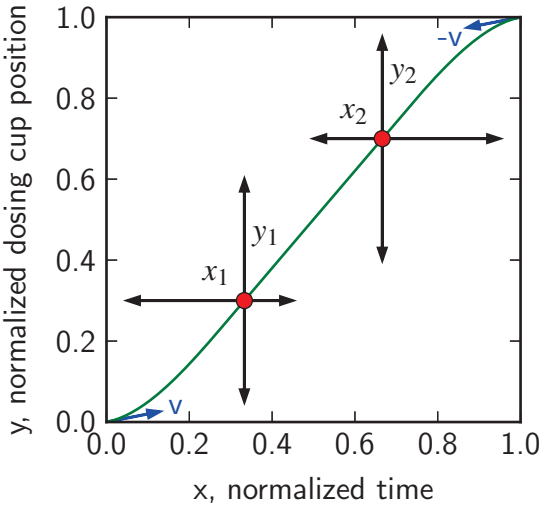

Figure 4: Motion degree of freedom. Red points represent the base points of the motion function. The black arrows visualize the domain of $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$.

cle a higher velocity to open the cup fast and subsequently reduce the cup's velocity to drop the rice in a defined rate. Afterwards, the cup must be accelerated to the starting velocity v for the next cycle.

The motion is described by a motion transfer function. It describes the normalized position of the dosing cup over the normalized time. So the starting point of the motion represents $x = 0.0$, where $x = 1.0$ marks the end time point of motion.

In this work the motion transfer function is modeled with five degrees of freedom, concatenated by piecewise polynomials. Section one and three are third order polynomials, the mid section is a straight line. Figure 4 shows one instance of the motion transfer function. The red basis points P_1 and P_2 can be shifted within the domain of black arrows. If $y_2 < y_1$ the cup moves back.

The periodic velocity v (blue arrow) assures the jerk-free transition of consecutive cycles. So a wide range of different functions can be created.

In principle it is possible to use higher order polynomials instead of third order polynomials to get high order derivatives. But for simplicity in this work only cubic polynomials are used.

2 Particle Simulation

Bulk materials like rice, coffee beans or pills play an important role in processing and packaging machines. The demand for faster machines requires new methods of machine design and optimization. In contrast to solids or fluids, there is no generally acknowledged theory which describes granular matter. Although the physics of granular matter are only based on mechanical interactions, the physical description proves very complex due to the large number of reaction partners.

Due to the development of more powerful computers, it is possible to model and simulate granular matter for understanding the basic mechanism of granular matter through numerical simulations. There are different methods for the simulation of particle systems. For some problems only the whole system behavior is in the scope of interest. These can be solved with continuum approaches [20].

One example of such a problem is the calculation of the wall tension in tanks or silos [14]. But the approach loses its validity if single particle effects play a major role. In this case, a particle based method like the discrete element method (DEM) can be used.

2.1 Discrete Element Method

In comparison to continuum-based approaches the DEM considers the behavior of individual particles. The trajectory and rotation of each particle are obtained by using a numerical time integration scheme. The calculation of the forces acting at a contact point of two colliding particles is done by applying suitable contact models [5, 23]. The forces are calculated from the overlap of two colliding particles with virtual spring and damper elements (figure 5). Newton's equation of motion is solved by numercal time integration. To reduce the computation time, sphere or multispheres are often used for modeling the shape of the real particle geometry. A good introduction in DEM modeling and simulation is given in [21].

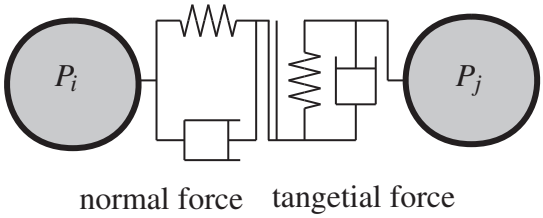

Figure 5: DEM contact model

2.2 Modeling and Simulation

This section describes how the problem is modeled and the DEM is implemented.

Scene Construction For simulation the open source DEM simulator Yade[4] is used in which a variety of constitutive laws are implemented. The core of Yade is written in C++. For scene construction it provides Python bindings. Yade can be used either interactively with 3D viewer and Python console or in batch mode.

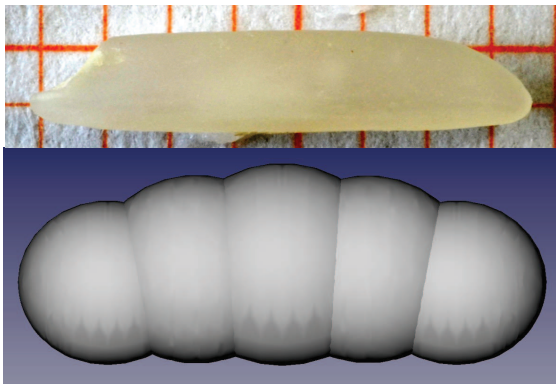

Figure 6: A Golden Sun Basmati rice grain and its DEM representation.

A model script for Yade is written in Python. Yade comes with functions for generating or loading sphere packings, materials or meshes. For post processing simulation traces can be exported in VTK[5] format or 3D rendering of the scene. In batch mode input files are specified to create non-interactive simulation runs. User-specific functions can be used for evaluation.

Rice Modeling The DEM approach can handle a lot of different particle types, e. g. spheres, ellipsoids, cylinders, superquadratics or polyhedrons. Often particles, especially spheres, can be composed to clumps with rigid connection or damped springs between the clump members. The advantage of spheres, compared with other particle representations, is the simplicity of detecting contacts between particles. Assuming two particles P_i and P_j with their radii r_i and r_j and positions $\vec{x}_i$ and $\vec{x}_j$ then equation 1 gives the distance $d_{surface}$ between the surface of two particles. If $d_{surface} \leq 0$ the particles are in

[4] https://yade-dem.org/

[5] http://www.vtk.org/

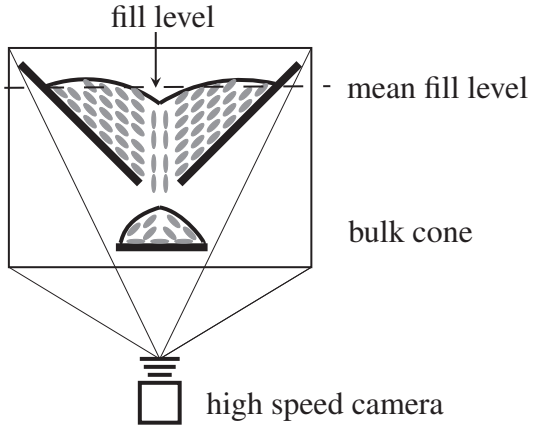

Figure 7: Calibration experiment setup

contact.

$$d_{surface} = \|\vec{x}_i - \vec{x}_j\| - r_i - r_j \qquad (1)$$

The contact detection of other particle shapes results in more complex equations and leads to higher computational costs.

Often granular materials do not have a spherical shape. Idealizing these materials with spheres can cause unexpected side effects during simulation. Therefore we clump spheres together to complex particle shapes. The size and shape of the spherical clump members are a compromise between model accuracy and computational efficiency.

DEM-Model Calibration Yade comes with a variety of different build-in constitutive laws including the well-known Cundall-Strack [5] constitutive law. Choosing one should be done really carefully to reproduce all relevant macroscopic effects of the granular material.

For finding appropriate material parameters we design a small experiment (figure 7) which (mostly) can be evaluated automatically and therefore does not require much user interaction. A transparent hopper is filled with $150\,g$ of Golden Sun Basmati rice. The rice flows out of the hopper and creates below the hopper a bulk cone. The mean fill level is tracked over time with an high speed camera.

Table 1: Material Parameter Steel Hopper and Rice

Parameter	Value	Unit
damping	$d = 0.125$	–
gravity	$g = (1+d)\cdot 9.81$	$\frac{m}{s^2}$
Steel Hopper		
Young's modulus	$E = 2e11$	Pa
Poisson's ratio	$\nu = 0.33$	–
density	$\rho = 7800$	$\frac{kg}{m^3}$
friction	$\mu = 25$	$^\circ$
Rice		
Young's modulus	$E = 1e8$	Pa
Poisson's ratio	$\nu = 0.2$	–
density	$\rho = 1592.67$	$\frac{kg}{m^3}$
friction	$\mu = 43$	$^\circ$

The experiment is also implemented in a DEM simulation. During a simulation it is possible to track also the fill level. Comparing the fill level time series from experiment and simulation gives one criterion to decide whether the simulation model is realistic. Other criteria are the angle of respose of the bulk cone or the shape of the fill level. But these criteria can not be tracked reliable so they have to be evaluated by hand.

To calibrate the model, we must define the particle shape and material parameter. The Cundall-Strack [5] constitutive law uses the Young modulus, Poisson's ratio and a friction value per material. A global damping parameter handles the restitution.

The particle shape especially the number, size and position of spheres within a clump affects the flow behavior of the granular material. If a particle is more spherical, it can better roll than one with a cylindrical shape.

The grooves between two spheres can cause an implicit source of friction. Particles can attach to each other or can get caught on edges.

In [19] the rice grain is modeled with 11 spheres with nearly original measures for length and width. But 11 spheres with a diameter between $1\,mm$ to $2\,mm$ per clump leads to a huge demand on computational effort. Therefore we studied clumps with different number of spheres per clump. With two or three spheres per clump the particle is to spherical. Despite of high friction values the flow out time in the simulation was significantly shorter than in the experiment. Therefore five spheres per clump is a good compromise between model detail and computational effort.

Figure 6 shows a comparison of a real rice grain and the DEM particle.

Clearly evaluated the influence of particle shape within a DEM simulation of rice in a ploughshare mixer [4]. He used spherical and superquadratic particles to model rice grains and ends up with a significant better model behavior with the more realistic superquadradic particles than the spherical ones.

The material parameter of the hopper are set according to table 2. Due to the small particle velocity along the hopper walls friction is the crucial material parameter.

In [19] some material parameter for rice are given with $E = 2e8\,Pa$, $\nu = 0.2$ and $\rho = 1574\,\frac{kg}{m^3}$. These values are used as a starting value for calibration.

The experiment video captured with the high speed camera (figure 7) is processed with MATLAB® Image Processing Toolbox™. So it is possible to extract the hopper mean fill level over time. This experiment was repeated five times and the time series averaged (see green plot figure 8).

The calibration of the DEM material parameters can be formulated as an minimization problem. The objective function is the summed squared point-wise differences of the mean fill level over time. The objective function has its minimum if both plots are equal. Hence, the experiments and simulations have the same fill level at every time step and the hoppers are empty at the same time point. In theory the objective function converges to zero but in practice always remains a finite value due to discrepancies of the model and the experiment.

The degrees of freedom of the optimization are friction, Young's modulus (only order of magnitude), Poisson's ratio of the rice particles, friction of the hopper and global damping. The optimization is implemented with

Table 2: Material Parameter of the transparent hopper.

Transparent Hopper		
Parameter	Value	Unit
Young's modulus	$E = 3e9$	Pa
Poisson's ratio	$\nu = 0.41$	–
density	$\rho = 1190$	$\frac{kg}{m^3}$
friction	$\mu = 20$	°

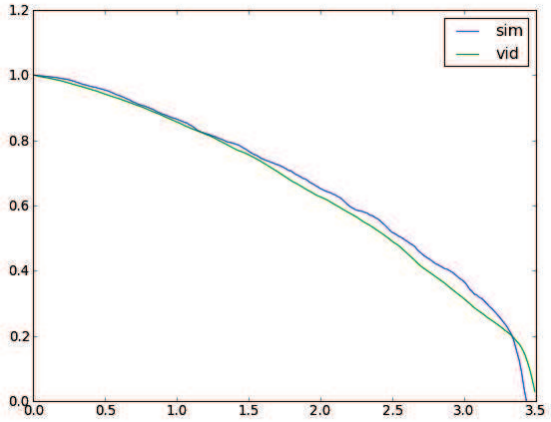

Figure 8: Mean fill height over time im simulation and experiment.

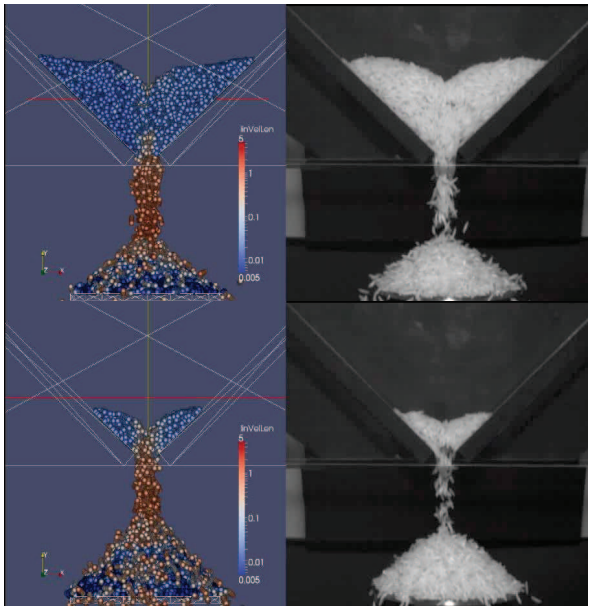

Figure 9: Rice flows out of a hopper, comparison between simulation and experiment

the SciPy[6] box-bounded L-BFGS implementation. For evaluating one parameter set one simulation has to be performed. This took about 10 hours on an Intel® Core™ i7-3770.

The result of the optimization is shown in figure 8. The green plot shows the mean fill level of the experiment, the blue plot shows the fill level of the simulation. As expected the plots are not equal, but quite similar.

Bridging, the bulk cone angle of repose and the shape of fill level surface can not be tracked and evaluated reliably with computers. Therefore, the effects are compared manually. Figure 9 shows two frames of the comparison video of the optimization result.

At this point the main part of the calibration is done. As the final step the DEM model has to be calibrated at the desired machine.

The hopper and the following tube are made from stainless steel, the hopper is sandblasted. At the end the model has to be calibrated for the steel parts. But the intransparency of steel disturbs the usage of cameras. Therefore, also a transparent hopper and tube are used. The machine runs with $90\,\frac{cycles}{min}$ and doses $1\,kg$ rice per cycle. A high speed video of the hopper section is captured and manually compared with the simulation. The video analysis with MATLAB® of this setup is difficult because there are no easily extractable characteristics. Some points which are compared are shape of rice falling through the hopper, the points in time when the first respectively last rice grain passes the hopper.

This experiment is used to fine tune the material parameters obtained from the optimization and validate these in a realistic scenario. Young's modulus and Poisson's ratio were correctly determined by the optimization. But friction and damping of all materials are sensitive to changes in scenario and particle speed. Therefore, a couple of combinations are tried out.

One simulation run with $1\,kg$ rice takes round about $30\,h$ on four cores[7]. The hopper opening in the machine is significant larger so scaling the particles to reduce the computational effort is a good choice.

In [3] the rice particle of figure 6 is scaled by a factor of 1.5 without significant lost of accuracy. The particle scaling reduces the time per simulation run to $12\,h$. Figure 10 shows the comparison of simulation and experiment with the final material parameters from table 2.

[6] `http://docs.scipy.org/doc/scipy-0.13.0/reference/generated/scipy.optimize.fmin_l_bfgs_b.html`

[7] The granular material is less dense while falling down so the Verlet algorithm can work more effective.

2.3 Numeric Noise

This section explain floating point issues by using DEM which influences the selection of optimizers but is not essential for understanding the main point of this work.

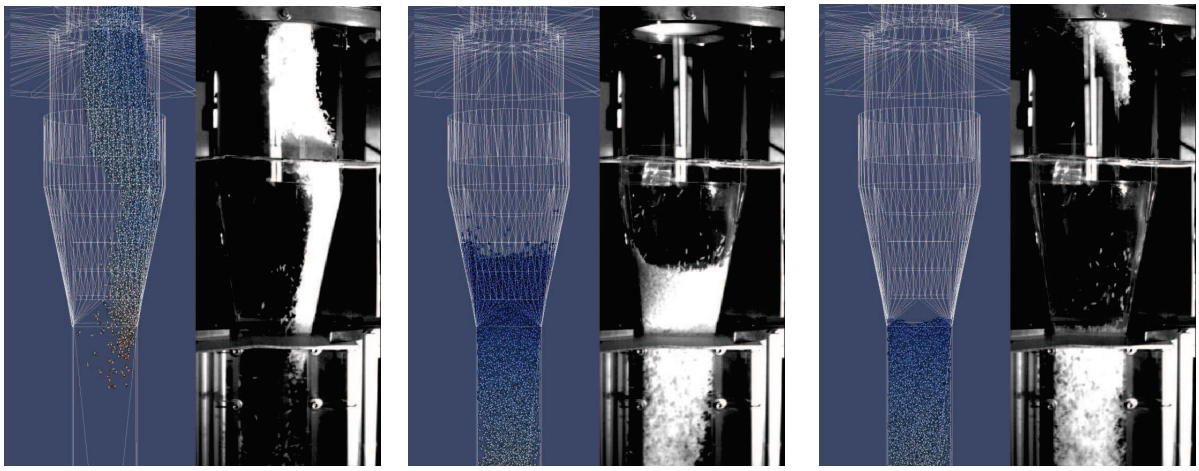

Figure 10: Rice flows out of the dosing cup through the hopper. Comparison between simulation and experiment

DEM simulation is a numeric method to solve differential equations. Typically, computers use floating point numbers to represent real numbers. But floating point data types are finite. A typical 64 bit `double` variable has approximately 16 decimal fractional digits[8]. Hence, numbers with more fractional digits are rounded. This leads to some difficulties. The main problem with respect to DEM is the invalidity of associativity and distributivity. This means these properties are valid for intuitive usage. The terms `(5.0 + 3.0) + 2.0` and `5.0 + (3.0 + 2.0)` result still in `10.0` Therefore associativity seems to be valid. But the small Python example in listing 1 shows a case where associativity does not hold.

Listing 1: error-prone terms for addition

```
1 a=(1./19.+ 1./24.)+1./17.
2 b= 1./19.+(1./24. +1./17.)
```

[8] `double`, with 53 bit mantissa, $53 \cdot log_{10}(2) \approx 16$

Pythons standard floating point type is `double`. So the results are:

$$a = 0.15312177502579\mathbf{977}$$
$$b = 0.1531217750257\mathbf{98}$$

and the corresponding difference is:

$$a - b = -2.7755575615628914\mathbf{e\text{-}17}$$

The error is some orders of magnitude smaller than the result's accuracy a user would expect. So within this example the error does not matter.

The error becomes critical if many operations are subsequently executed and the error is accumulated. If a sequence of operations is executed in the same order the error is always the same and maybe undetected because every time the algorithms results in the same value.

Often numeric programs are parallelizable. According to associativity, it does not matter which order of summation is used to calculate

the result. Thus, some threads can calculate partial result and finally reduce them to a final result[9]. The examples tell us associativity does not hold every time but the used algorithms uses this fact to speed up performance.

So if a numerical algorithm is executed, the result is error-prone, hopefully in a small order of magnitude. If the algorithm is executed single-threaded, the error is always the same. But with multi-threaded execution the operation system schedules the threads in quasi random order. Because of this execution order of the floating point operations, the result is different between some executions. Hence, the algorithm becomes non-deterministic.

Regarding to DEM simulation, the floating point issue is problematic because DEM is an iterative algorithm. Hence, every iteration step is executed with the noisy data of its predecessor. With increasing iteration number, the error grows step-by-step. This leads to small displacements of the particles over time. Often there are no macroscopic effects noticeable. If a simulation run is evaluated manually, small inaccuracies are not conspicuous. Some instabilities within the simulation are often fixed by reducing the time step. But this do not solve the problem in general. Smaller time steps increase the stability of the algorithm, but also increases accumulated rounding errors. More details about floating point arithmetics can be found in [9, 7].

Optimization with DEM-Models The problem occurs if a multi-threaded DEM simulation is used to derive an objective function for optimization. Through the high number of steps[10] the error can be noticeable. In this work, the error is nearly half a order of magnitude of the objective function. It depends on CPU architecture, time step size and number of particles. A pair of error-prone objective functions values are in some situations not clearly decidable whether it is a good or bad one. Assuming a minimization problem the bad objective value, biased with a negative error can exceed the good one with a positive error. So an optimizer would choose the bad one for generating the next iteration. A Quasi-Newton optimizer would have serious difficulties to determine correct gradients. If the noise has a similar order of magnitude like the objective, the step size for determining the gradient has to be large. Otherwise the gradient is noisy and misleads the optimizer. However, large step sizes increase the risk of step over an extrema.

This problem is minor if the (metaheuristic) optimizer explores the search space. If the individuals are distributed over the whole search space, a big difference in objective values can be assumed. In this phase the differences in objective values are greater than uncertainty of error-prone evaluations. So the optimizer can decide which are good and bad individuals.

In the phase of exploitation, typically the differences of objective values are smaller. So it is harder to decide between the individuals. There are some strategies to handle the uncertainty. The simplest approach is to run replications for every individual and average the resulting objective values. This would burn a large additional amount of computing power but leads to more reliable result.

In [8] an approach is given to adapt the calculation accuracy during optimization[11]. For numerical algorithms this means changing the floating point data type, e. g. from `single`, `double` to `long double`. So the precision of calculation is increased and the uncertainty shifted to a smaller significance level. Using

[9] Adding up large series of floats is also problematic because of cancellation. See [24, 16]

[10] 10^5 to 10^6 per simulation run

[11] This article is related to Discrete Event Simulation.

more and smaller particles does not help at this point. This would increase the model detail but not the numerical stability. Therefore the data types must be changed and recompiled within the simulator. So this approach causes additional code maintenance. The data types of Yade can be changed via a macro during compilation.

In [12] an adaptive method is given to reevaluate individuals of a population rankwise. Individuals are sorted by their objective value and reevaluated. The method assumes the uncertainty is too large if their rank changes at this point. It is also possible to reevaluate only the better individuals of a population because only these are relevant to generate the offspring.

Another approach is to let the uncertainty unconsidered. In phase of exploration it is not so important because the decision is often correct [8]. Also, the expected number of details of the optimization must also be defined. If we expect (nearly) perfect solutions uncertainty handling is unavoidable. But if good candidate solutions are sufficient, then uncertainty handling is not necessary because exploitation or manual fine-tuning is done by the user.

3 Optimization

The optimizer is the central part to find good candidate solutions of the formulated problem. There is a huge amount of different optimization strategies. They can be divided in local and global ones. Local ones, e.g. greedy algorithms or L-BFGS using gradients to choose the search direction with the best benefit. Strategies of this class typically converges really fast to extrema, but often it is a local one. To find global extrema the algorithm must be restarted several times at different points in the search space. But there is no guarantee to find it. Objective functions based on simulations typically do not have known derivatives. Hence, their gradients must be approximated which can negatively influence the performance or the quality of the results. Is the objective noisy determining a gradient can be difficult or expensive (see section 2.3).

The other class are global optimization strategies. Widely used algorithms are simulated annealing, particle swarm optimization or some variants of genetic algorithms or evolution strategies. Often this class of algorithms is robust to multitude of local extrema, uncertainty or strange constraints. The main disadvantage of this algorithm class is there is no proof about convergence or correctness.

All of these optimizers have one point in common: they do not guarantee to find the global extrema.

The problem described in this work is defined in a box-bounded continuous domain. With 13 degrees of freedom there is no reliable assertion about presence or amount of local extrema. For this reason, the problem has to be treated as black-box with noisy objective values.

3.1 Constraints and Penalty Terms

Constraints are search space restriction of the optimization problem. They can be distinguished into result requirements and restrictions to candidate solutions. They must be distinguished from the bounds of search space which only limits the problem to a more feasible problem size. If solutions were found at the search space limits, the bounds were defined adverse.

Restrictions Hard constraints are conditions to candidate solutions which must never be violated. Especially for using a real ex-

periment instead of a simulation the test station could break down. So the optimizer must not enter these forbidden regions of search spaces.

Requirements Soft constraints are requirements to the final result, e.g. the dosing cup should be empty after filling the bag. But if the cup is not empty no model or test bed breaks, it is only a non optimal candidate solution.

Hard constraints can limit the performance or success of the optimizer. If the optimizer searches in region of search space which border to a hard constraint then the optimizer have to do one large step[12], which requires the possibility of adapting the step size, to negotiate the forbidden region. If this is not possible the optimizer converges prematurely.

By using soft constraints these specific regions are only undesired. So the optimizer can visit these regions but gets only badly evaluated individuals. So it is possible to go through a region of search space with soft constraints with smaller steps and reach on the other side a maybe better region [6]. Soft constraints are modelled with penalty terms. In doing so, penalty points are added to an objective value to cover undesired system behaviour.

3.2 Objective Function

In section 1.1 a description of the problem was given. In this section the aim of this work is formulated into an objective function.

The aim is to speed up the filling time, i. e. the time which the rice needs to pass the end of the filling tube. So the naive approach is to measure the time between the first and the last particle. Because of numeric noise (see section 2.3) it is inconvenient to model the whole system behaviour dependent on only two particles. Numeric noise slightly changes the particle position or velocity on every iteration. In the end the particle positions respectively velocities are error-prone in an order of magnitude which makes the objective value useless for optimization.

Therefore, a more robust approach is used here. If a particle passes the end of the filling tube, a time stamp is inserted in the list `listOfTimestamps`. At the end of the simulation the standard deviation is calculated (equation (4)) over `listOfTimestamps`. When the rice needs a long time to fall down, the many different time stamps appear in the list. Hence, the standard deviation is large. If the rice falls down compactly, there are less different time stamps and the standard deviation is small. So the objective value is not dependent only on two but on all particles.

$$l_c = 0.05 \cdot \texttt{leftParticlesInCup()} \quad (2)$$

$$l_h = 0.5 \cdot \texttt{leftParticlesInHopper()} \quad (3)$$

$$g = \texttt{std(listOfTimestamps[0:-20])} \quad (4)$$

$$f = g + l_c + l_h \quad (5)$$

In some rare cases one particle sticks on edges of the hopper. If this happens to the last one, the objective value is significantly enlarged in spite of ”good” model behavior. So the last 20 particles are excluded from applying the objective function see equation (4)).

According to section 3.1 only soft constraints are used. Equation (2) counts the number of particles which remain in the cup at the end of a simulation run. The dosing cup is not empty so the volume in the bag is smaller than desired.

Equation (3) counts the number of particles which are in the hopper at the simulation end point. Especially in the early explo-

[12] Some smaller step is forbidden because they would violate the hard constraint.

ration phase, when the optimizer tests some crazy geometries, some particles remain in the hopper because the particles are too slow or stopped or time is over. These particles are not measured at the filling tube end. Therefore, equation (4) does not depend on these particles. Equation (4) is also not affected by equation (2) because these particles left the cup. The factors in equation (2) and (3) are weighting factors to scale the impact of the terms.

The final objective function is given in equation (5). The g term models the desired fast bag filling. This term is minimized in the exploitation phase. The penalty terms l_c and l_h exclude some undesired system behavior or strange hopper geometries. During the exploration phase these penalty terms should approach to zero.

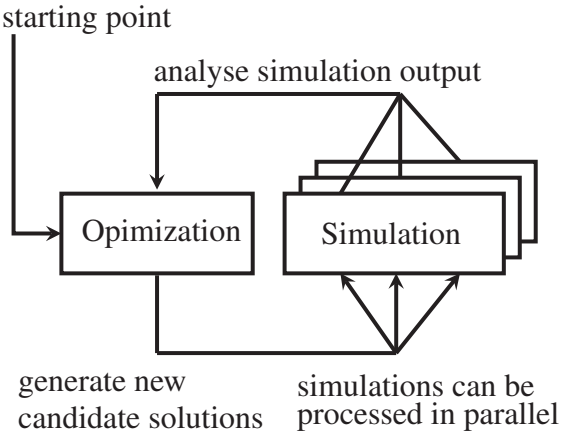

Figure 11: Simulation-based Optimization

3.3 Choosing an Optimizer

No Free Lunch Theorem The No Free Lunch theorem (NFL) for optimization[13] states in simple words there is no universal perfect optimization strategy which is good for all classes of optimization problems [25]. So if one algorithm outperforms another in some classes of problems, the other way round this algorithm is badly suited for some other problem classes. This means algorithms to solve the Traveling Salesman Problem very fast and with really good results may not be convenient for this work. A good introduction for understanding NFL is given in [13].

Covariance Matrix Adaption Evolution Strategy In this work there is a global optimizer needed for a box bounded continuous domain with a small number of degrees of freedom with noisy objective values. The difficulty with Quasi-Newton optimizers with noisy objectives was explained and discussed in section 2.3. Therefore, this class of algorithms is excluded.

There is a lot of literature about global optimizers. A good introduction in metaheuristics is given in [18].

Here the Covariance Matrix Adaption Evolution Strategy (CMA-ES) by Hansen [11] is used. CMA-ES is a derandomized evolution strategy which approximates the covariance matrix of the objective function. It is well suited for ill-conditioned, non-seperable, multimodal, non-convex or noisy functions. It is easy to use because only one strategy parameter, the initial step size `sigma0`, has to be adjusted by the user. The main point to select CMA-ES is performance. Hansen [10] shows that CMA-ES is often significantly better than other algorithms in that field. There are some variants for CMA-ES with some restarting capabilities [1], multi objective functionality [15] or surrogate model support [17].

[13] The name is derived from the idiom "There ain't no such thing as a free lunch." [22].

3.4 Implementation

3.4.1 Parallelization

The CMA-ES uses the concept of population. Good parent individuals are selected and the (hopefully better) offspring are added to the population. Typically, there are no dependencies between the individuals. Also, the order of evaluation is irrelevant. So the population concept opens the door for distributed simulation[14] by using desktop grids or clusters (see figure 11). For an efficient job scheduling it is favorable to set the population size to a small multiple of available processors to avoid idle time caused by non-optimal scheduling.

Table 3: CPU-times during optimization

No. of Particles	Generations	Population size	CPU-time per individual (standard deviation σ)	CPU-time per generation
4600	1 - 196	16	$344.4\,s$ ($128.1 \triangleq 36.3\%$)	$1.53\,h$
4600	197 - 274	24	$241.8\,s$ ($12.5 \triangleq 5.2\%$)	$1.61\,h$
6700	275 - 305	24	$712.3\,s$ ($47.6 \triangleq 6.7\%$)	$4.75\,h$
18380	306 - 315 316 - 328 [15]	24	$7365.3\,s$ ($375.2 \triangleq 5.1\%$)	$49.10\,h$

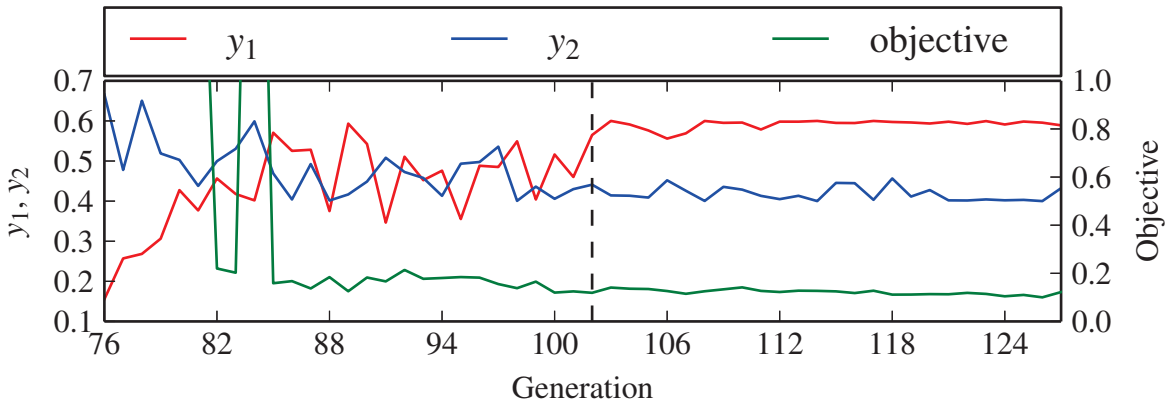

Figure 12: y_1 and y_2 values of each best individual of iteration 76 to 127

One simulation run can use parallel simulation[16]. Yade uses OpenMP for parallelization but its speed up is not linear.

[14] Different processors work on different simulation runs.

[15] Filling time as objective used

[16] Different processors work on one simulation run.

3.5 Running the Optimization

Simulation-based optimization has a high demand on computation, often some thousand

CPU hours. A fast optimization algorithm is mandatory to reduce computational effort. The idea is to simplify the model to get coarse informations about the model with less effort. In the exploration phase, the diversity of objective values is high. Although the individuals are not exactly evaluated their comparability is hopefully still preserved. So the optimizer during exploration phase can run many iteration with significantly reduced computational effort. With decreasing diversity of objective values within one population the accuracy must be increased to preserve comparability between individuals (see [8]).

In this work, this approach is implemented by scaling the particle number, as a consequence also particle size. The main characteristics with less particles are still given but exact metrics like filling times can not be determined accurately with a coarse model. Switching between accuracy levels is still an open question. In [8] is the switching done depending on the slope of the objective function. In the present case it is done manually.

The optimization was executed on an Ubuntu 14.04 LTS machine with Intel® Core™ i7-3770 and 8 GB main memory. Table 3 show the used number of particles during optimization. At the beginning of the optimization the execution times and their standard deviation are huge. In this period the geometry and motion are inefficient so particles need a long time to pass the hopper. Later, when the optimizer finds better candidate solutions, the execution time is much shorter. The model exits prematurely because all particles have passed the hopper. The simulation domain of the model is empty, and therefore, it stops.

This optimization (generation 1 to 315) consumed 1063.7 CPU hours[17]. Without particle size scaling the estimated CPU-time would be $315 \cdot 49.1h = 15467h$.

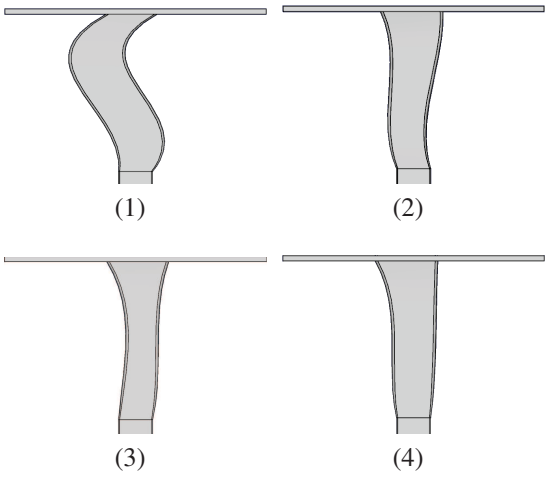

Figure 13: Some examples of geometries occurring during optimization.

Generation 1 to 315 is executed with the objective function (5). But there is no proof that this function will result in good candidate solutions. Some good individuals are found, but maybe some better ones could be found by using another objective function. Therefore, from generation 316 filling time is used as anobjective function.

Figure 15 shows the overall progress of the optimization. For each generation the best objective value is given. The dashed vertical lines mark restarts of the optimizer. Note, the number of particles can only be changed at a restart. Some slightly different objective values occur which can negatively influence the learned Covariance Matrix of the CMA-ES. Although disadvantageous learned informations can be corrected, this process may take longer than restart and relearn. This can also have a positive side effect for exploration because unvisited regions may be found. Further information is given in [1].

The optimizer tries various kinds of hopper shape. Due to the fact that a black box optimizer does not know the problem, it can not skip curious candidate solution which a hu-

[17] 1 CPU hour ≙ 1 hour on 1 core

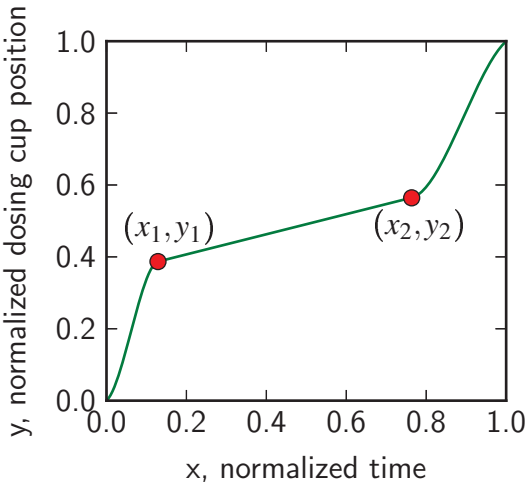

Figure 14: Motion transfer function of the best solution found

man designer might do. Some examples are given in figure 13.

Trapped in a local Minimum As mentioned in section 1.3 the optimizer visit a region of search space where yield $y_1 > y_2$. So the dosing cup will move backwards. This situation could be avoided by using a penalty term. But falsely determining such a term can mislead the optimization into a wrong direction.

From iteration 102 (figure 12) the optimizer is trapped in a local minimum with non optimal motion. The minimal found objective value was 0.10009484673 so far. From iteration 102 to iteration 127, 98.7 % of individuals have a y_1 greater than y_2. The remaining individuals were not able to escape from the local minimum. So the optimization was stopped manually and restarted from the point with minimal fitness so far. The values of y_1 and y_2 are exchanged[18] here.

[18] A genetic algorithm can do this by recombination.

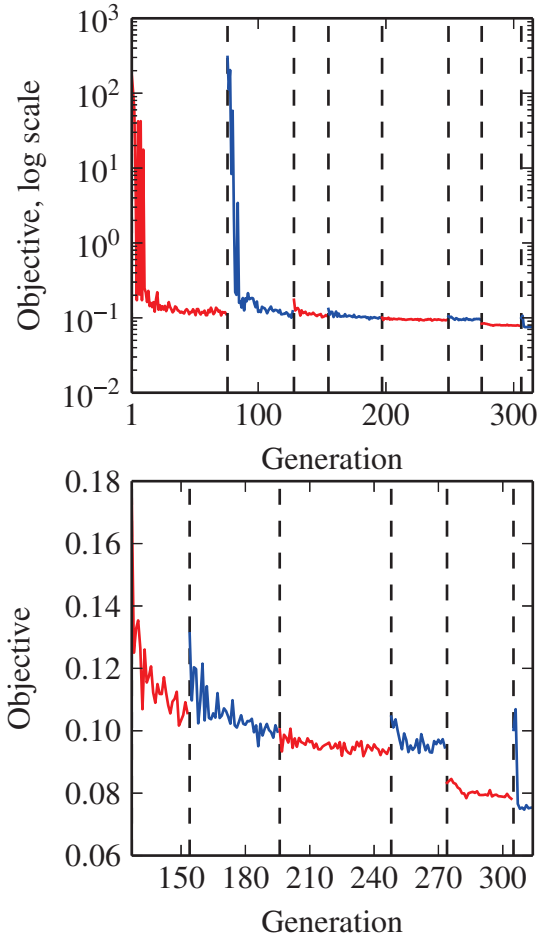

Figure 15: Progress of optimization. Vertical dashed lines and also color change represents optimizer restarts.

CMA-ES has some build-in heuristic convergence criteria to detect premature convergence. This is useful for a fully automated run. But these criteria often needs some more iterations to detect premature convergence than an expert user.

3.6 Evaluation of Results

Table 4 shows the best found objective values of the optimization. There is only a positive correlation between the objective function (5) and filling time. The reason might be that the

Table 4: Best solutions found

Found in generation	objective function	Filling time
313	0.07592	$0.4128\,s$
324 [15]	0.07442	$0.4138\,s$
320 [15]	0.07619	$0.4188\,s$
311	0.07626	$0.4218\,s$
311	0.07584	$0.4218\,s$
309	0.07500	$0.4318\,s$

filling time could influence the particle distribution in the hopper.

To check if the optimizer can find better solutions by using the filling time as an objective, the optimizer is restarted at the best found solution so far. This is done in generation 316-328 (see table 3), but no better solutions were found.

Table 4 shows that the more robust objective function (5) also leads to an acceptable solution. Figure 16 shows the simulation of the final result. Figure 14 shows the final motion. In the first phase of motion the cup accelerates to open the cup quickly. Then the cup moves with a constant velocity over the dropping hole to define the particle amount within the hopper. In the third phase the cup is empty, so it can be moved away fast.

The solutions of table 4 are quite similar in geometry and motion. The small differences are due to numeric noise of the simulation. Therefore it is not necessary to let the optimizer search for better solution because the optimizer can not decide between a better solution and only better objective values.

The best found solution has a filling time of $413\,ms$. The aimed filling time from section 1.1 is $450\,ms$. So there is some surplus time to increase process safety or machine speed.

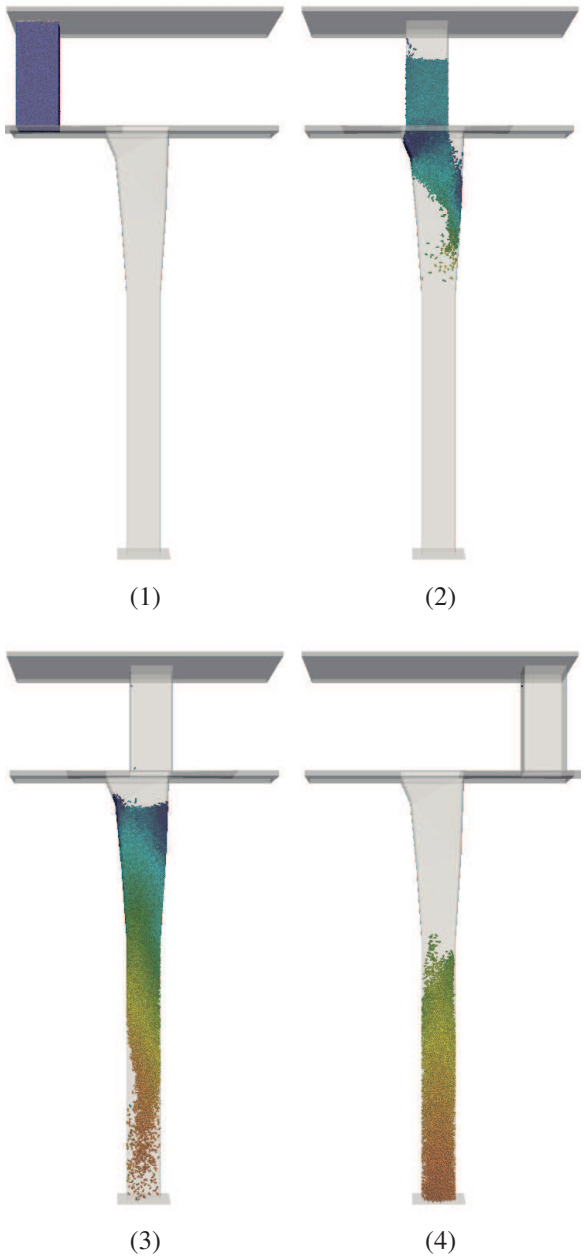

Figure 16: Simulation of best found solution at different times. Particles are colored by its velocity.

References

[1] Anne Auger and Nikolaus Hansen. A restart CMA evolution strategy with increasing population size. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pages 1769–1776. IEEE, 2005.

[2] Chair of Processing Machines and Processing Technology. Datenbank vat. `http://tu-dresden.de/die_tu_dresden/fakultaeten/fakultaet_maschinenwesen/ifvlv/vat/mlumw/index_html?id=2017`, awailable only in German.

[3] Georg Clauß. Entwicklung einer Strategie zur Geometrieoptimierung mittels der Diskreten-Elemente-Methode. Diploma thesis, TU Dresden, Institut für Verarbeitungsmaschinen und Mobile Arbeitsmaschinen, 2013.

[4] Paul W Cleary. Particulate mixing in a plough share mixer using DEM with realistic shaped particles. *Powder Technology*, 248:103–120, 2013.

[5] Peter A. Cundall and Otto D. L. Strack. A discrete numerical model for granular assemblies. *Geotechnique*, 29(1):47–65, 1979.

[6] Gunter Dueck. *Das Sintflutprinzip*. Springer, 2004.

[7] Fernández, José-Jesús and García, Inmaculada and Garzón, Ester M. Floating point arithmetic teaching for computational science. *Future Generation Computer Systems*, 19(8):1321–1334, 2003.

[8] Matthias Frank, Christoph Laroque, and Tobias Uhlig. Reducing computation time in simulation-based optimization of manufacturing systems. In *Proceedings of the 2013 Winter Simulation Conference*, pages 2710–2721. IEEE, 2013.

[9] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys (CSUR)*, 23(1):5–48, 1991.

[10] Nikolaus Hansen. The CMA evolution strategy: a comparing review. In *Towards a new evolutionary computation*, pages 75–102. Springer, 2006.

[11] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[12] Hansen, Nikolaus and Niederberger, André SP and Guzzella, Lino and Koumoutsakos, Petros. A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. *Evolutionary Computation*, 13(1):180–197, 2009.

[13] Yu-Chi Ho and David L Pepyne. Simple explanation of the no free lunch theorem of optimization. In *Proceedings of the 40th IEEE Conference on Decision and Control, 2001*, volume 5, pages 4409–4414. IEEE, 2001.

[14] J Mark FG Holst, Jin Y Ooi, J Michael Rotter, and Graham H Rong. Numerical modeling of silo filling. I: continuum analyses. *Journal of engineering mechanics*, 125(1):94–103, 1999.

[15] Christian Igel, Nikolaus Hansen, and Stefan Roth. Covariance matrix adaptation for multi-objective optimization. *Evolutionary computation*, 15(1):1–28, 2007.

[16] W. Kahan. Pracniques: Further Remarks on Reducing Truncation Errors. *Commun. ACM*, 8(1):40–, January 1965.

[17] Ilya Loshchilov, Marc Schoenauer, and Michéle Sebag. Bi-population CMA-ES agorithms with surrogate models and line searches. In *Proceeding of the fifteenth annual conference companion on Genetic and evolutionary computation conference companion*, pages 1177–1184. ACM, 2013.

[18] Sean Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013. Available for free at http://cs.gmu.edu/~sean/book/metaheuristics/.

[19] Darius Markauskas and Rimantas Kacianauskas. Investigation of rice grain flow by multi-sphere particle model with rolling resistance. *Granular Matter*, 13(2):143–148, 2011.

[20] D Niedziela, S Schmidt, K Steiner, J Zausch, and C Zemerli. Continuum numerical simulation of multiphase granular suspension flow in the context of applications for the mechanical processing industry. *International Journal of Mineral Processing*, 2015.

[21] Thorsten Pöschel and Thomas Schwager. *Computational granular dynamics*. Springer, 2005.

[22] William Safire. On Language; Words Left Out in the Cold. *New York Times*, Feb 14., 1993.

[23] J Shäfer, S Dippel, and DE Wolf. Force schemes in simulations of granular materials. *Journal de physique I*, 6(1):5–20, 1996.

[24] Jonathan Richard Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry*, 18(3):305–363, 1997.

[25] David H Wolpert and William G Macready. No free lunch theorems for optimization. *Evolutionary Computation*, 1(1):67–82, 1997.