Developing an Open Python Library for Urban Design Optimisation - Pyliburo

Kian Wee Chen[1], Leslie Norford[2]
[1]CENSAM, Singapore-MIT Alliance for Research and Technology, Singapore
[2]Department of Architecture, Massachusetts Institute of Technology, USA

Abstract

Urban design optimisation is a powerful method for the exploration of multiple designs. In performing an urban design optimisation, we need to link and automatically execute multiple domain-specific applications, a technically complicated setup. Current solutions resolve the technical obstacle by embedding the applications within a single Computer-Aided Design (CAD) application to streamline the setup. The solution leverages the CAD application's modelling workflow and capability to process the urban geometries for analyses. However, this solution is workflow specific; users either do not have access to optimisation algorithms or are restricted to the capabilities provided by a specific CAD application. For optimisation to be accessible to a wider community, we develop an open Python library, Pyliburo, to provide optimisation capability to all design workflows. Pyliburo aims to be easily integrated into a user's existing design workflow to provide or enhance optimisation capability. To do so, Pyliburo emphasises interoperability, platform independence, ease of use, integration flexibility and extensibility. There are many ways Pyliburo can be integrated into an existing workflow, and we will show a particular workflow in detail as a case study to demonstrate its capability.

Introduction

It is common in the urban design process to explore multiple designs and their impact on urban performance metrics. One powerful way of facilitating the exploration is using an optimisation algorithm. However, setting up an urban design optimisation is a complicated technical process, in which the setup needs to link many domain specific applications and automate their execution. Currently, the solution to overcome this technical obstacle is embedding the optimisation process within a Computer-Aided Design (CAD) application such as Rhinoceros3D Grasshopper (Koenig and Varoudis (2016); Taleb and Musleh (2015)), Generative Components (Mueller and Strobbe (2013); Turrin et al. (2012)), Dynamo Revit (Rahmani Asl et al. (2015)) and Houdini3D (Kaushik and Janssen (2013)). The solution requires users to construct the 3D model in the CAD application; the geometries from the 3D model are processed and passed to domain-specific applications for analyses, which are usually linked to the CAD application as plug-ins. The analysis results are then used in the optimisation algorithm as feedback to generate better designs. The process is streamlined as it is orchestrated within a single interface. The solution leverages the CAD's modelling workflow and capability to process the urban geometries for analyses. The disadvantage is the optimisation setup is workflow specific. Depending on the CAD application, either the users do not have access to optimisation capability, or they are restricted to the optimisation capabilities of the specific CAD application.

For a wider community to have access to optimisation capability, we propose an open Python library, named Pyliburo (Python Library for Urban Optimisation) that provides or enhances optimisation capability for all design workflows. Pyliburo complements the existing ecology of optimisation workflows by aiming to be readily integrated into a user's existing design workflow to either 1) provide optimisation capability or 2) enhance the optimisation capability. Modular and developed in Python, Pyliburo emphasises interoperability, platform independence, ease of use, integration flexibility and extensibility:

- **Interoperability** – Pyliburo adopts an open data model, CityGML (Gröger and Plümer (2012)) for data exchange to streamline the linking and execution of applications in the optimisation process.
- **Platform independence** – Pyliburo is independent of any CAD application and Operating System (OS). It uses a powerful geometry kernel, PythonOCC, for processing urban geometries and it is portable to all OS environments.
- **Ease of use** – Pyliburo is targeted at urban designers with basic programming skills so that they can use the library to provide or enhance optimisation capability to their existing workflow. Thus, the library has a syntax that is accessible to amateur programmers (Perez et al. (2011)).
- **Integration flexibility** – Pyliburo needs to be readily integrated into existing workflows. Python is effective in interfacing with different applications and programming languages for integration (Perez et al. (2011)).
- **Extensibility** – Pyliburo needs to be extensible for designers to develop customised algorithms for their individual purpose. Python has a comprehensive set of libraries (Dubois and Dubois (2007)) for developing and extending Pyliburo.

CPlan is an existing open C# library with a similar objective (Koenig (2015)). However, CPlan only has a rudimentary geometry library for constructing basic geometrical shapes. CPlan also does not support any open data model. Although CPlan is open and extensible, the lack of existing open libraries in C# makes it difficult to

extend CPlan. In comparison, Python has an extensive set of open libraries, which is evident in the development of Pyliburo and the case study demonstration. We utilise many existing libraries to extend Pyliburo in both cases.

The next section will explain the core packages that are in Pyliburo and the potential use cases. The following section illustrates in detail one of the use cases in a case study. The case study integrates Pyliburo into an existing workflow to demonstrate Pyliburo's capabilities. Lastly, we draw conclusions and discusses future improvements for the library.

Core packages and use cases of Pyliburo

Pyliburo has three independent core packages. Designers can use the packages as a whole or take the packages apart to set up an optimisation process. The three packages provide the vital functions for extending and integrating the library into existing workflows.

Py3dmodel processes all the geometrical calculations. It is based on PythonOCC (PythonOCC (2016)), a Python wrap of the industrial-grade geometric modelling kernel OpenCascade. It is often difficult for researchers to navigate through PythonOCC's extensive Application Programming Interface (API) to find the right function for the modelling task at hand. Py3dmodel is a wrap of PythonOCC that exposes only the essential functions for modelling 3D urban design. Examples of these functions include geometry construction (extrusion, loft and booleans), measurements (surface area, edge length and the distance between geometries), modification (translate, rotate and scale) and deconstruction of geometries (retrieving points from the polygon, edges from the polygon and polygons from the solid). Py3dmodel enables Pyliburo to be independent of CAD applications.

Pycitygml reads and writes CityGML, an XML-based open data model, and is developed using the lxml library (lxml (2016)). Pycitygml provides an easy-to-use API for designers to read and write CityGML objects such as building, land-use and road. Applications in the optimisation process use CityGML as the data exchange format.

Pyoptimise consists of the optimisation algorithms available for urban design optimisation and exploration. Pyoptimise also includes data analytics and visualisation algorithms for researchers to explore the result of the optimisation. Currently, we have only implemented the Non-Dominating Sorting Genetic Algorithm II (NSGAII) for optimisation, Pareto-ranking for analysing the optimisation results and lastly a scatter plot for visualisation in an easy-to-use API.

With the three core packages, Pyliburo can be adapted for varying use cases. The use cases include but are not limited to providing optimisation capability to existing non-optimisation workflow, providing evaluation methods for an existing optimisation workflow, providing optimisation algorithm for an existing optimisation workflow and linking multiple existing workflows for an optimisation process.

Use case 1 – providing optimisation capability to existing non-optimisation workflow

Pyliburo can be adapted to provide optimisation capability to existing non-optimisation workflows. Figure 1 illustrates this adaptation, where the existing workflow is a non-parametric 3D application. In an optimisation process, a parametric model is commonly used for representing the design problem. Parametric modelling represents a design as a set of parameters (Aish and Woodbury (2005)), either independent or derived from other parameters. Independent parameters are the input parameters and are constrained within a limited range. By varying the input parameters, the parametric model generates multiple design variants. The optimisation algorithm is linked to the parametric model, whereby the algorithm varies the input parameters to generate better performing design variants according to the performance objectives.

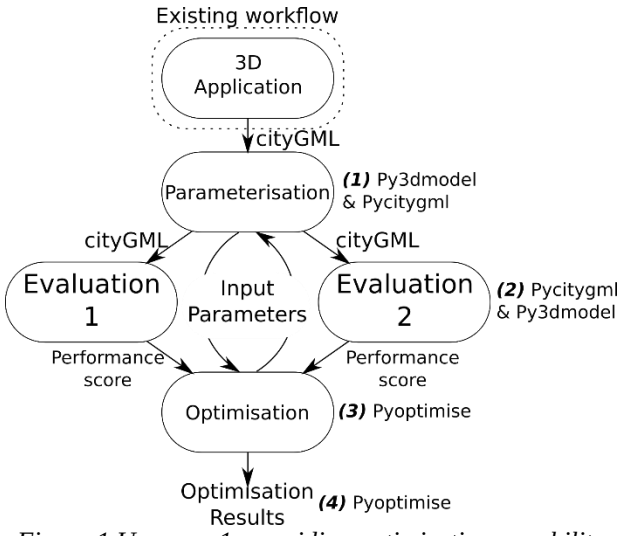

Figure 1 Use case 1: providing optimisation capability for an existing non-optimisation workflow

1. The non-parametric 3D application exports a CityGML model of the design. Pycitygml will read the CityGML model, and Py3dmodel will set up the parametric model. The parametric model will generate multiple new design variants, and Pycitygml writes the design variants into CityGML.
2. Pycitygml reads the CityGML design variants and passes the geometries to Py3dmodel. Py3dmodel will analyse the geometries according to the performance objectives or process the geometries for analysis by an external simulation application such as Radiance/Daysim or EnergyPlus.
3. Pyoptimise uses the input parameters and performance scores of the design variants for optimisation. The optimisation algorithm generates new and stronger design variants and

passes the input parameters to the parametric model in (1). The cycle continues as specified by the designers.

4. As we are unaware of any open standard format for documenting results from optimisation, we use an XML file with a customised schema. The documented data include the design variant's identity, generation, input parameters and performance score. Pyoptimise provides functions to read, write, analyse and visualise the XML file.

Use case 2 – providing evaluation method for an existing optimisation workflow

Pyliburo can be adapted for use in an existing optimisation workflow to extend the workflow evaluation capability. Figure 2 illustrates the adaptation, where the existing optimisation workflow could be Rhinoceros 3D grasshopper and requires one of the evaluations in Pyliburo for its optimisation process. Instead of re-implementing the evaluation in grasshopper, it can easily interface with Pyliburo by exporting a CityGML file.

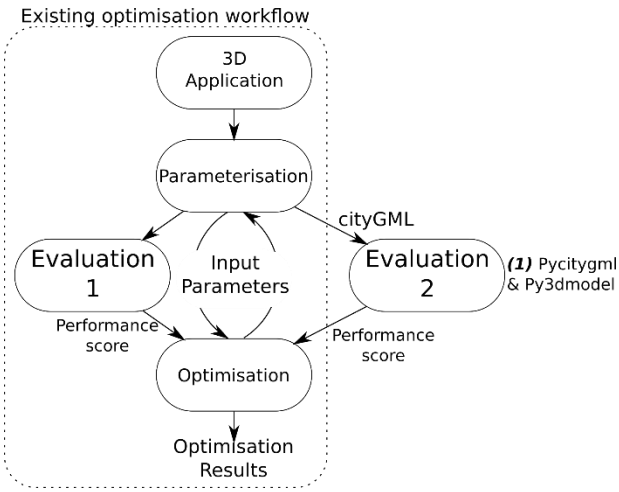

Figure 2 Use case 2: providing evaluation method to an existing optimisation workflow

1. The existing workflow needs to export a CityGML model of the design variant. Pycitygml then reads the CityGML design variants and passes the geometries to Py3dmodel. Py3dmodel will process the geometries for analysis by an external simulation application and returns the performance score in the appropriate form into the optimisation algorithm in the existing workflow.

Use case 3 – providing optimisation algorithm for an existing optimisation workflow

Pyliburo can be adapted to provide an optimisation algorithm for an existing optimisation workflow. Figure 3 illustrates the adaptation, where the existing optimisation workflow requires the optimisation algorithm in Pyliburo.

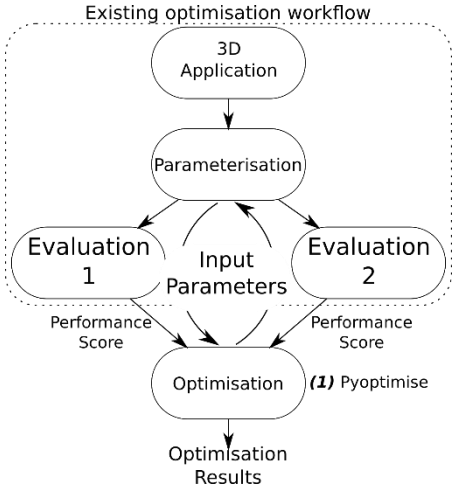

Figure 3 Use case 3: providing optimisation algorithm for an existing optimisation workflow

1. The existing optimisation workflow needs to export and import the customised XML schema used by Pyoptimise to have access to the optimisation algorithm. Pyoptimise provides an API for reading and writing the customised XML schema. The link can be established by extracting and returning the input parameters and performance scores from the existing workflow through a text file.

Use case 4 – linking multiple workflows for an optimisation process

Pyliburo can be adapted to link multiple workflows for an optimisation process, as shown in Figure 4. In this use case, designers will fully utilise the existing ecology of workflows and use Pyliburo as a workflow management system for pushing data from one application to another. A possible scenario is when the designer uses SketchUp for 3D modelling, Urban Modelling Interface (UMI) (Reinhart et al. (2013)) for urban daylight, CitySIM (Robinson et al. (2009)) for city energy simulation and GenOpt (Wetter (2001)) for optimisation.

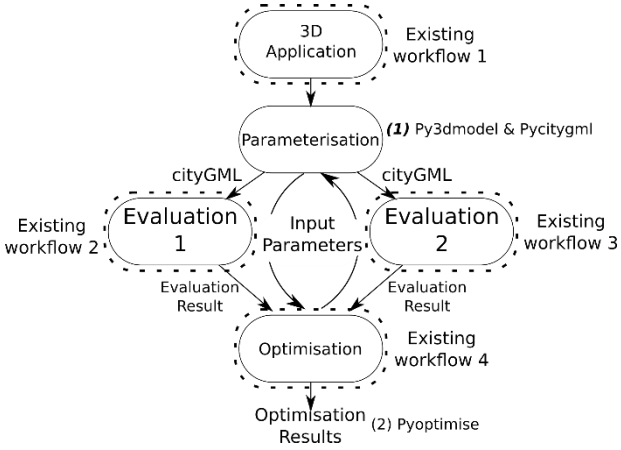

Figure 4 Use case 4: linking multiple workflows for an optimisation process

1. Existing workflow 1 (non-parametric 3D application) needs to export a CityGML model for Pyliburo. Pyliburo will then parameterise the

design as in Figure 1(1). CityGML is used as the main data exchange format for all the evaluations (existing workflow 2 & 3). If existing workflows 2 & 3 do not readily accept CityGML for simulation, the data can be processed using Py3dmodel and Pycitygml into the appropriate format for the workflows. Pyliburo will sort the input parameters and performance scores into the right format for existing workflow 4 (optimisation) and retrieve the result accordingly.

Case Study

The case study illustrates use-case 1 in detail, in which Pyliburo is used to provide optimisation capability to an existing non-optimisation workflow. Designers equipped with basic programming skills can use Pyliburo to set up the optimisation process as shown in the case study and, more broadly, will have complete flexibility in setting up the optimisation process for individual projects.

The case study is a simplified scenario located in Singapore and consists of 25 tower blocks arranged in a five-by-five grid as shown in Figure 5. We want to explore the impact of different configurations of the buildings regarding their height, orientation and positioning on the density and daylight performances. Density and daylight are contradictory performance objectives. The increase in density will result in taller blocks, which leads to inter-shading between blocks and decreases the daylight performance. From the optimisation process, we will obtain design variants with varying trade-offs between the two performance objectives.

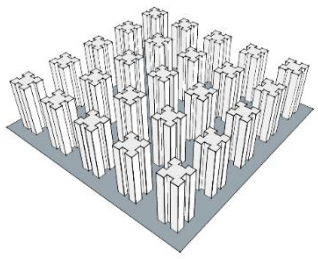

Figure 5 Case study with maximum density

Retrieving geometries for parameterisation

The case study is conceived and modelled in Sketchup. Sketchup's intuitive modelling workflow makes it very popular, but it is a non-parametric 3D modelling application with no existing workflow to model a parametric model.

In this case, we can reconstruct and parameterise the model in Figure 5 using the Py3dmodel package of Pyliburo, but this is not desirable as it is time-consuming. Instead, we will use the 3D model constructed in SketchUp as a skeletal model for parameterising the design and reduce the required time and effort. Figure 6 shows the workflow.

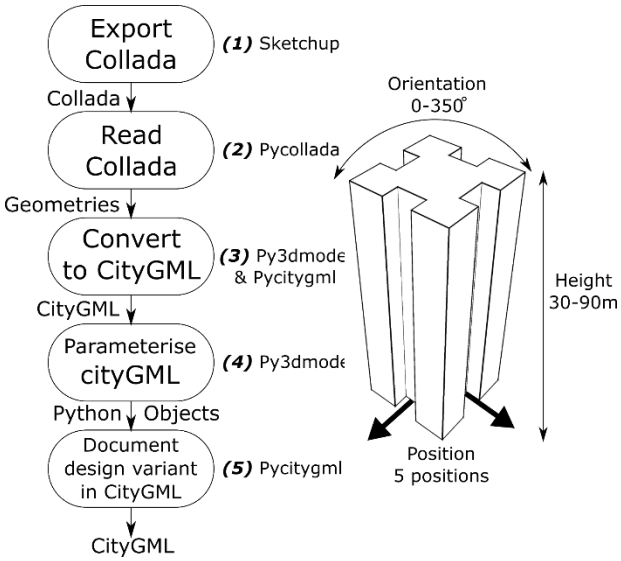

Figure 6 (left) Parameterisation workflow (right) Parameters of a building

1. Sketchup exports the geometries in Collada format (Collada (2016)). Collada is an open data model for documenting geometries and does not have city semantics.
2. We use an existing Python library, Pycollada (Pycollada (2016)), to read the Collada file.
3. We use Py3dmodel to process meshes from Collada into solids and polygons and sort them into respective CityGML building and land-use objects. Pycitygml then writes the processed geometries to CityGML format.
4. Py3dmodel uses the CityGML as a skeletal model for the parameterisation. Base on our intention, we parameterise each building into three parameters: height, orientation and position.
    a. The building is scaled in the Z-axis according to the height, *H*, parameter, where 90m>*H*>30m, 3 | *H*.
    b. The building is moved according to the position, *P*, parameter, which has five possible positions: original position or move 10m in either north, south, east and west direction, where 0<=*P*<=4.
    c. The building is then rotated anti-clockwise about its centre point according to the orientation, *O*, parameter, where 0<=*O*<=350º, 10 | *O*.
    d. The parametric model has 75 parameters in total.
5. Pycitygml writes each generated design variant into CityGML format as shown in Figure 7.

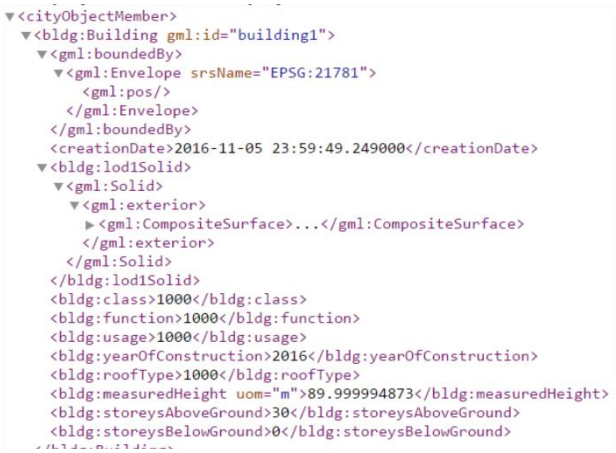

Figure 7 A snippet of a CityGML design variant

Density evaluation

We measure the density objective using Floor Area Ratio (FAR), which is the ratio of total floor area to the plot area. Figure 8 shows the workflow.

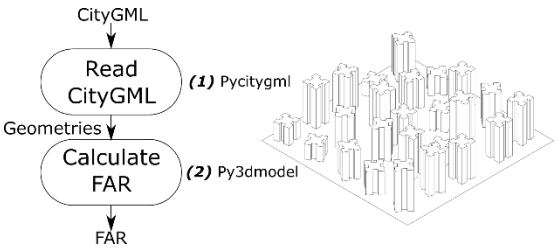

Figure 8 (left) Density evaluation workflow (right) Design variant with a FAR of 3.76

1. Pycitygml reads the design variant documented in CityGML.
2. Py3dmodel measures the plot area, footprint area and height of each building, then based on the floor-to-floor height of 3m, calculates the number of levels of a building. Multiplying the footprint area by the number of levels gives the floor area of a building. Summing the floor area of all the buildings gives the total floor area. Dividing total floor area by plot area gives the FAR.

Coupling with Radiance for daylight evaluation

We measure the daylight performance using the Daylight Façade Area Index (DFAI) (Compagnon (2004)), which is the fraction of the façade area receiving annual mean illuminance (lx) equal to or greater than a threshold value. The annual mean illuminance threshold value is about 10,000lx in the tropical climate (A.L. Martins et al. (2014)). A 10,000lx threshold value for the façade will result in 500lx annual mean illuminance on the interior work plane (Compagnon (2004)). We extend Pyliburo to couple with the GenCumulativeSky module (Robinson and Stone (2004)), distributed with Daysim, to compute the annual cumulative illuminance. We then divide the annual illuminance by the daylight hours provided by the weather file to get the annual mean illuminance. Figure 9 shows the workflow.

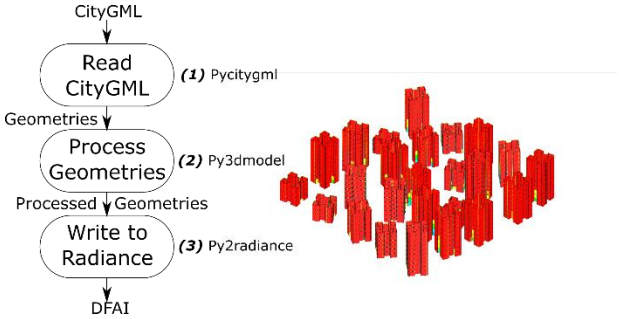

Figure 9 (left) DFAI evaluation workflow (right) Design variant with a DFAI of 90.7%, red represents surfaces achieving the threshold value.

1. Pycitygml reads the design variant documented in CityGML.
2. Py3dmodel processes and sorts the geometries into Radiance/Daysim ready geometries.
3. We use an existing Python library, Py2radiance (Janssen et al. (2011)), to write and execute the GenCumulativeSky analysis and calculate the DFAI.

Optimisation workflow

Once the design problem and evaluations have been defined and set up, we link all the procedures with the optimisation algorithm. We use NSGAII for the optimisation process as the genetic algorithm is one of the most common and effective optimisation algorithms used in the built environment field (Evins (2013)). Figure 10 shows the workflow.

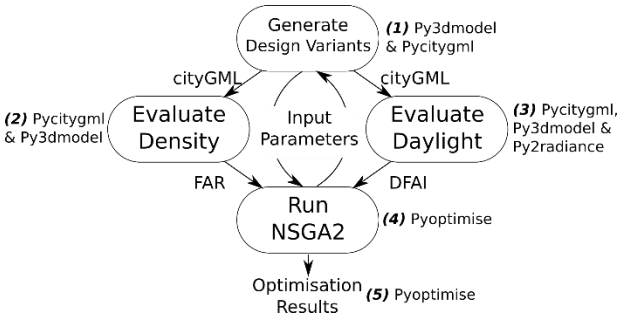

Figure 10 Optimisation workflow

1. We set up the parametric model using Py3dmodel as shown in Figure 6. With the input parameters, the parametric model generates new design variants. Pycitygml writes the design variants into CityGML.
2. Pycitygml reads the CityGML design variants and passes the geometries to Py3dmodel. Py3dmodel calculates their FAR according to their geometries.
3. Pycitygml reads the CityGML design variants and passes the geometries to Py3dmodel. Py3dmodel prepares the geometries for input into Radiance. Py2radiance writes the Radiance inputs and execute GenCumulativeSky. The DFAI is calculated using the Radiance results.

4. Pyoptimise uses the input parameters and performance scores of the design variants for reproduction in the NSGAII algorithm. NSGAII performs reproduction using one-point crossover according to a crossover rate of 0.8, and each new design variant is subjected to mutation according to a mutation rate of 0.01. Each generation produces a new and stronger population of design variants and passes the input parameters to the parametric model in (1). The cycle continues as specified by the designers.
5. Pyoptimise writes the optimisation results into a customised XML schema as shown in Figure 11. Each design variant is assigned an identity, the generation it is reproduced in, the status indicating if it is alive or dead in the NSGAII environment, input parameters and its performance scores.

```
▼<data>
  ▼<population>
    ▼<individual>
       <identity>0</identity>
       <generation>0</generation>
       <status>false</status>
     ▶<inputparams>...</inputparams>
       <derivedparams/>
     ▼<scores>
         <score>3.218</score>
         <score>91.472</score>
       </scores>
     </individual>
```

Figure 11 Snippet of the customised XML schema used in Pyoptimise

Results

We ran the optimisation for 50 generations and generated 5000 design variants. There are 51 design variants on the Pareto-front as shown in Figure 12.

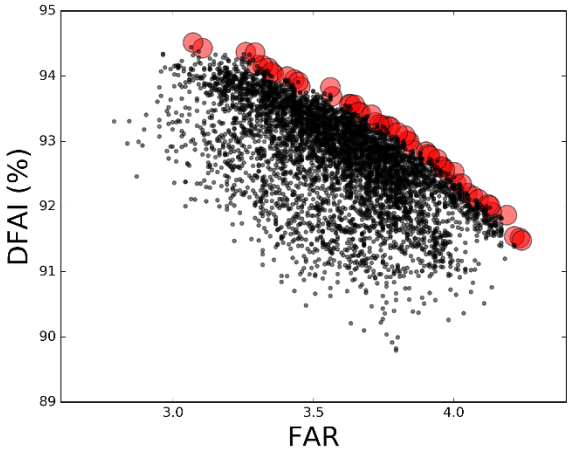

Figure 12 The optimisation generated 5000 design variants with 51 on the Pareto-front (red dots)

Figure 13 shows three design variants with the same density but different DFAI performance. By varying the building heights, design variant 4942 achieves the best DFAI of the three, 93%. The shorter blocks are placed at strategic locations in between taller tower blocks to reduce the amount of inter-shading between tall towers. In comparison to design variant 4942, the heights of the tower blocks of design variant 338 are more uniform, which result in inter-shading between blocks of similar heights and reduction in DFAI by 2.5%. Design variant 1222 has a DFAI of 91.5%; the tower blocks are of similar heights with a few shorter blocks placed in between, but not as strategically placed as design variant 4942. Through these three examples, we can see the optimisation algorithm's search for optimal design variants throughout the generations.

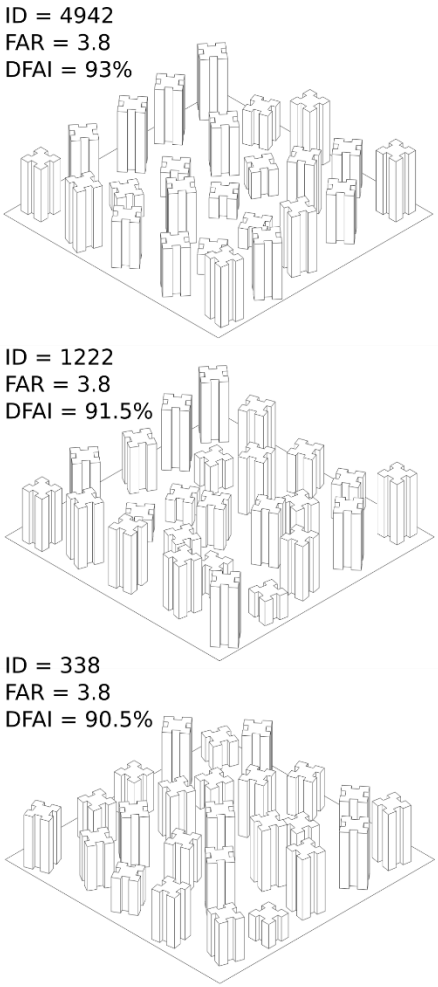

Figure 13 Three design variants with same density but different DFAI performance

Lastly, the designers can explore the design variants by setting certain criteria. In this case, we took a cross section of the design variants at FAR 3.8 and compare the good design variant on the Pareto-front (design variant 4942) and the bad design (design variant 338). This process can be repeated for other FAR or DFAI. Through exploring the design variants, it is possible for designers to derive design strategies, which in this case is to position shorter blocks at strategic locations between taller tower blocks.

Conclusion

This paper presented an open Python library, Pyliburo, developed to support urban design optimisation for all design workflows. We showed how Pyliburo could

provide optimisation capability to an existing non-optimisation workflow in the case study. In this study, we retrieved geometries from Sketchup using an open data model called Collada, converted Collada to CityGML, parameterised the CityGML model, extended Pyliburo's capability by implementing a customised algorithm for calculating FAR and integrating Radiance as its evaluation for the optimisation using py3dmodel, Pycitygml and existing Python packages. All these were possible because Pyliburo has a powerful geometric modelling kernel, Py3dmodel, to process the urban geometries and adopts the use of open data model for data exchange in its workflow. We then successfully set up and ran an optimisation process using the optimisation package, Pyoptimise, and the results provided feedback for the design problem.

There are three main areas for improvements:

1. Reduce the computation time for the optimisation. Currently, a 50 generations NSGA2 optimisation takes about 100 hours, which is too slow to provide timely feedback for the researchers. We propose two methods to speed up the process. Firstly, we propose to integrate Pyliburo with a parallel computing platform so that we can bring the optimisation onto the cloud. Secondly, use a small population optimisation algorithm, micro-GA, which uses a lower computational power.
2. Provide a more comprehensive set of evaluations for the performance objectives. We plan to integrate more evaluation methods such as solar accessibility analyses to assess the potential for photovoltaics panels and solar heat gain through the façade, frontal area index to assess urban ventilation and connectivity to assess road network design.
3. Interpret the optimisation result for supporting decision-making. The optimisation result consists of thousands of design variants. Even after Pareto-ranking and filtering, there will remain many design variants. The extraction of knowledge to support decision and selection of design variants for further design development are not a straightforward process. Thus, we propose to use advanced techniques from the field of multi-criteria decision analysis to facilitate the interpretation process. These techniques include clustering, archetypal analysis and the use of S-Pareto front.

Pyliburo is still in its early development stage; we encourage interested individuals to contribute packages and modules to the library or extend its capability by coupling it with other existing Python libraries. For example, users can couple it with such other optimisation libraries as PyOpt or DEAP to assess a diverse range of optimisation algorithms. Eventually, we would like to develop a graphic interface for Pyliburo so that it is accessible to a wide community of designers. Without a friendly interface like Rhinoceros Grasshopper, only advanced users with programming background will use Pyliburo. However, visual programming interfaces like Grasshopper have their disadvantages. The visual network gets complicated with the increase in nodes and iterations. We require further investigation into developing a better interface for setting up a design optimisation process. Nevertheless, we envision that Pyliburo, accompanied by a graphic interface and supported by workflow examples and documentation, will make performance-based urban design optimisation more accessible to urban designers.

Acknowledgement

This research was supported by the National Research Foundation Singapore through the Singapore MIT Alliance for Research and Technology's Center for Environmental Sensing and Modeling interdisciplinary research program.

References

Aish, R., Woodbury, R., (2005). Multi-level Interaction in Parametric Design, in: Butz, A., Fisher, B., Krüger, A., Olivier, P. (Eds.), *Smart Graphics*, *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 151–162.

A.L. Martins, T., Adolphe, L., E.G. Bastos, L., (2014). From solar constraints to urban design opportunities: Optimization of built form typologies in a Brazilian tropical city. *Energy Build.* 76, 43–56.

Collada, (2016). Khronos Group: Connecting Software to Silicon [WWW Document]. URL https://www.khronos.org/collada/ (accessed 10.27.16).

Compagnon, R., (2004). Solar and daylight availability in the urban fabric. *Proc. Int. Conf. Sol. Energy Build. CISBAT 2001* 36, 321–328.

Dubois, P.F., Dubois, P.F., (2007). Guest Editor's Introduction: Python: Batteries Included. *Comput. Sci. Eng.* 9, 7–9.

Evins, R., (2013). A review of computational optimisation methods applied to sustainable building design. *Renew. Sustain. Energy Rev.* 22, 230–245.

Gröger, G., Plümer, L., (2012). CityGML – Interoperable semantic 3D city models. *ISPRS J. Photogramm. Remote Sens.* 71, 12–33.

Janssen, P., Chen, K.W., Basol, C., (2011). Iterative Virtual Prototyping: Performance Based Design Exploration, in: *29th eCAADe Conference Proceedings*. Ljubljana, Slovenia, 253–260.

Kaushik, V., Janssen, P., (2013). An Evolutionary Design Process – Adaptive-Iterative Explorations in Computational Embryogenesis, in: *Open Systems: Proceedings of the 18th International Conference on Computer-Aided Architectural Design Research in Asia*. Singapore, 137–146.

Koenig, R., (2015). CPlan - An Open Source Library for Computational Analysis and Synthesis, in:

Proceedings of the 33rd eCAADe Conference. Vienna, Austria, 245–250.

Koenig, R., Varoudis, T., (2016). Spatial Optimisations - Merging depthmapX, spatial graph networks and evolutionary design in Grasshopper, in: *Proceedings of the 34th eCAADe Conference*. Oulu, Finland, 249–254.

lxml, (2016). lxml [WWW Document]. URL http://lxml.de/ (accessed 4.13.16).

Mueller, V., Strobbe, T., (2013). Cloud-based design analysis and optimization framework, in: *Proceedings of the 31st eCAADe Conference*. Delft, Netherlands, 185–194.

Perez, F., Granger, B.E., Hunter, J.D., (2011). Python: An Ecosystem for Scientific Computing. *Comput. Sci. Eng.* 13, 13–21.

Pycollada, (2016). Pycollada [WWW Document]. URL https://github.com/pycollada/pycollada (accessed 10.27.16).

PythonOCC, (2016). PythonOCC [WWW Document]. URL http://www.Pythonocc.org/ (accessed 4.13.16).

Rahmani Asl, M., Zarrinmehr, S., Bergin, M., Yan, W., (2015). BPOpt: A framework for BIM-based performance optimization. *Energy Build.* 108, 401–412.

Reinhart, C.F., Dogan, T., Jakubiec, J.A., Rakha, T., Sang, A., (2013). UMI - An Urban Simulation Environment for Building Energy Use, Daylighting and Walkability, in: *Proceedings of BS2013*. Chambéry, France, 476–483.

Robinson, D., Haldi, F., Kampf, J., Leroux, P., Perez, D., Rasheed, A., Wilke, U., (2009). CitySIM: Comprehensive Micro-Simulation of Resource Flows for Sustainable Urban Planning, in: *Proceedings for IBPSA 2009*. Glasgow, Scotland, 1083–1090.

Robinson, D., Stone, A., (2004). Irradiation modelling made simple: the cumulative sky approach and its applications, in: *Plea2004 - The 21st Conference on Passive and Low Energy Architecture*. Eindhoven, Netherlands.

Taleb, H., Musleh, M.A., (2015). Applying urban parametric design optimisation processes to a hot climate: Case study of the UAE. *Sustain. Cities Soc.* 14, 236–253.

Turrin, M., von Buelow, P., Kilian, A., Stouffs, R., (2012). Performative skins for passive climatic comfort: A parametric design process. *Autom. Constr.* 22, 36–50.

Wetter, M., (2001). GenOpt - A generic optimization program, in: Lamberts, R., Negrao, C.O.., Hensen, J. (Eds.), *Proceedings of the 7th IBPSA Conference, Volume I*. Rio de Janeiro, 601–608.