Use jEPlus as an efficient building design optimisation tool

Yi Zhang PhD MASHRAE
Institute of Energy and Sustainable Development,
De Montfort University, Leicester, United Kingdom
yizhang@dmu.ac.uk

Abstract
jEPlus is an open source tool originally developed for managing complex parametric simulations using EnergyPlus (E+). If coupled with optimisation algorithms such as Evolutionary Algorithms (EAs), it provides a convenient and highly efficient way to perform optimisation for building design and operation. This paper discusses the general processes of optimisation, and how jEPlus being used for simplifying the tasks of setting up optimisation for building simulation users. An optimisation problem with a solution space of more than one million design options is presented as a case study. EAs is coupled with jEPlus using the methods described in the paper, to perform both multi-objective and single objective optimisation. It is found that the multi-objective approach can produce better results than single objective methods.

Keywords jEPlus, EnergyPlus, Building Optimisation, Evolutionary Algorithms

1.0 Introduction
Building design optimisation is an important and growing research area, which has attracted attentions from more and more practitioners, too. In recent years, many optimisation exercises related to building design have been reported by researchers [1][2][3][4][5], most of who developed own tools that have not been made available to the community. One exception is GenOpt [1], a generic optimisation tool incorporating a range of algorithms and the ability to work with many building simulation programs.

Despite that general-purpose optimisation tools such as GenOpt have been available more than a decade, many researchers learned optimisation by developing their own algorithms. One of the main reasons is that optimisation problems involved in building design and operation vary vastly in nature, whereas there is not a ‘generic’ algorithm that is suitable for all problem types. To solve a problem effectively, researchers have to master the optimisation techniques, often by the means of implementing their own algorithms.

Learning optimisation techniques and developing algorithms is an enjoyable and rewarding job, especially nowadays it no longer requires writing code from scratch. Instead, readymade building blocks of algorithms can be customized and fitted together within a solver framework; the performance of the optimisation system can then be tuned by adjusting a handful of parameters. Involved in this process, however, are a few unavoidable tasks that are both tedious and error-prone. Coupling the optimisation system with a building simulation program is one of such tasks. A new tool is needed to fill in exactly this gap.

Coupling of an optimisation system with a building simulation program involves three steps: 1) prepare a simulation job of a solution to be evaluated; 2) start the simulation program with the job and wait until it finishes; and 3) collect results or error reports generated by the simulation program. Since optimisation always involves evaluation of a number of alternative solutions, being able to run simulations in parallel will be useful for accelerating optimisation process with modern computer hardware. In this paper, we describe the use of jEPlus [6] for optimisation and its advantages.

2.0 Optimisation problems

In general, an optimisation problem can be expressed in the following form:

Minimise: $\boldsymbol{J} = \boldsymbol{F}(\boldsymbol{X})$
Subject to: $\boldsymbol{G}(\boldsymbol{X}) \leq \boldsymbol{0}$
$\boldsymbol{H}(\boldsymbol{X}) = \boldsymbol{0}$
Where: $\boldsymbol{X} = [x_1, x_2, \dots, x_k]$ is the vector of problem variables;
$\boldsymbol{F}(\boldsymbol{X}) = [f_1(\boldsymbol{X}), f_2(\boldsymbol{X}), \dots, f_m(\boldsymbol{X})]$ is the vector of objective functions;
$\boldsymbol{G}(\boldsymbol{X}) = [g_1(\boldsymbol{X}), g_2(\boldsymbol{X}), \dots, g_n(\boldsymbol{X})]$ is the vector of inequality constraints;
$\boldsymbol{H}(\boldsymbol{X}) = [h_1(\boldsymbol{X}), h_2(\boldsymbol{X}), \dots, h_p(\boldsymbol{X})]$ is the vector of equality constraints.

Note that, mathematically, objectives, equality and inequality constraints are interchangeable. For example, an equality constraint $h(\boldsymbol{X}) = 0$ can be expressed as an inequality constraint by introducing a small tolerance ε, i.e. $g(\boldsymbol{X}) = |h(\boldsymbol{X})| - \varepsilon \leq 0$; or a constraint can be satisfied by minimising it as an objective function. Multiple constraints can be combined into one, e.g. $\hat{g}(\boldsymbol{X}) = \sum \boldsymbol{G}(\boldsymbol{X}) \leq 0$; so are the objective functions.

In real world applications, however, how to define an optimisation problem depends on the nature of the problem it tries to solve. For optimisation problems in building design and operation, objectives and constraints are often determined by user's priorities. For example, a building design can be optimised to minimize energy consumption (objective) while maintaining required comfort level (constraint); or it can be optimised to maximize comfort level (objective) within a limited budget (constraint), or indeed to minimise energy consumption, carbon emission, life cycle cost, discomfort, and more (objectives) all at once. It seems users should have complete freedom in choosing the form of the optimisation problem to meet their requirement. Unfortunately, their options are limited by the optimisation algorithms they are going to use.

Over the years, hundreds of optimisation algorithms have been developed. Broadly, they belong to three groups: gradient-based methods, pattern search and other non-population-based direct search methods, and population-based (and mostly, stochastic) methods. Only the last group of algorithms are capable of handling multi-objective and/or multi-constraint (often-called multi-criteria) problems. Other methods require objectives and constraints being combined into one criterion. In the case study of this paper, we will explain the difference between single and multi-objective approaches. Before that, let us look at one of the most popular optimisation algorithms of all times.

3.0 Evolutionary algorithms

Evolutionary algorithms (EAs) are a class of algorithms inspired by the Darwinian evolution theory. If evolution is the way in which nature created sophisticated beings like ourselves from single-celled organisms, using it to improve an engineering

design must be a trivial task. The concept of EAs is very simple: we start with a random set (population) of solutions, and then repeatedly evaluate the solutions and select better ones for creating new variants, until enough suitable solutions have been found or we have run out of time. The general processes of EAs[1] are illustrated in Figure 1.

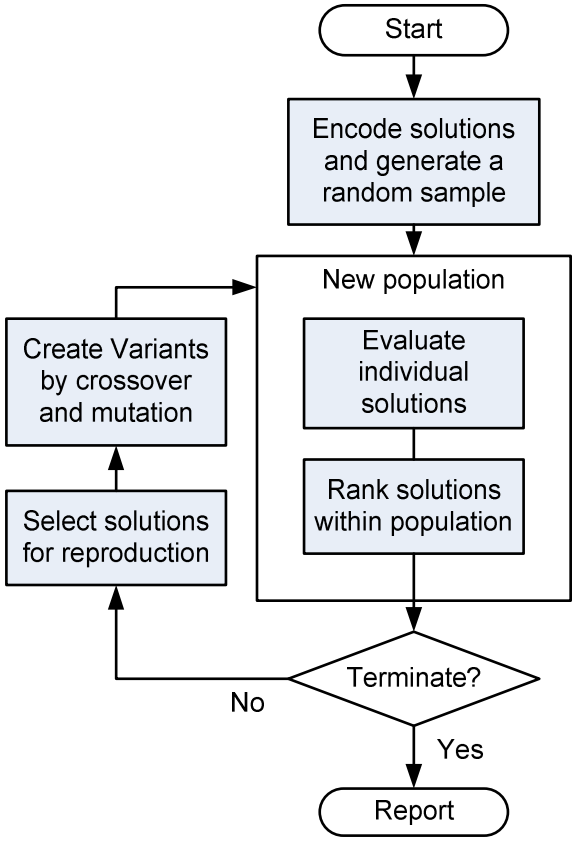

Figure 1 – The flowchart of EA processes

The processes start with implementing an encoding scheme for representing the solutions (the problem variables) in a numerical way that it can be handled by EAs. An encoding of a solution is called a 'chromosome', in which each variable is encoded as a 'gene'. Typically, the initial population of solutions is randomly sampled across the solution space. Each solution in the population will be evaluated for its 'fitness' for a given set of criteria. The ranking step works in conjunction with the selection procedure to ensure that the 'fitter' solutions will have a better chance to be used in making new variants. For most multi-objective and constraint handling methods, the ranking procedure encapsulates all the tricks. Crossover and mutation are nature-inspired ways for creating new solutions from existing 'parents'. Encoding-specific crossovers and mutation operators are often necessary for solving an optimisation problem efficiently. Normally, each optimisation project involves designing an encoding scheme, implementing solution evaluation method, selecting a suitable ranking scheme, and designing problem-specific crossover and mutation operators.

There are many available frameworks and tools for EAs, for instance, ECJ[2] in Java,

[1] There are several distinct flavours of EAs, such as Genetic Algorithm, Genetic Programming, Evolutionary Strategy, Evolutionary Programming, and more. In this paper we do not differentiate any particular algorithm, rather refer to them as a general method. Interested readers can find further information on individual algorithms in the literature.

[2] ECJ, see http://cs.gmu.edu/~eclab/projects/ecj/

OpenBeagle[3] in C++, and the global optimisation toolbox[4] in Matlab® to name just a few. Although it is reasonably easy to start using EAs with these tools, the first real hurdle a user will encounter is how to make them work with building simulation models. This, in fact, includes three questions: how to encode a building design problem in 'genes', how to express a new solution (in genes) as a building simulation model, and how to execute the simulation and collect results. jEPlus is offered to facilitate these three steps.

4.0 Solution encoding schemes

Three encoding schemes are commonly used in EAs. They are binary encoding, integer encoding, and real-valued encoding. Real-valued encoding not only represents a solution using the problem variables' native values, continuous or discrete, it can also capture the relationship between variables by imposing a data structure (such as a tree in the case of Genetic Programming). It however requires more consideration in designing problem-specific crossover and mutation operators to work with the encoding.

Integer encoding discretizes the continuous variables into finite and indexed values. It then represents a solution with indices of values of all problem variables. By discretizing continuous variables, integer encoding effectively reduces the solution space of the problem from infinite to finite. The disadvantage is good solutions may be lost due to discretization.

Binary encoding also discretizes the solution space, though implicitly. Binary form of the values of all discrete or continuous variables are put together in a bit string (a string of 0s and 1s) as the representation of a solution. For example, a double-precision value can be encoded in 64 bit (8 x 8 bytes), although more often a shorter encoding (e.g. 10 bit) is used to reduce the length of binary chromosomes. EAs with binary encoding can use a standard set of crossover and mutation operators, disregarding the problem it is solving.

For optimisation problems in building design and operation, most problems have both discrete and continuous variables. However, the precision requirement for continuous variables in buildings is very low. Take the width of shading device for example: a value 1.1152m instead of either 1.1m or 1.2m does not offer much benefit due to the presence of engineering errors and other uncertainties. In this case, integer encoding with predetermined design options may speed up optimisation by reducing the size of solution space. It is a native encoding scheme supported by jEPlus.

5.0 jEPlus project and integer chromosome

jEPlus was originally introduced in 2009 as a parametric tool for EnergyPlus[5]. It allows users to define parameters within an EnergyPlus model, and manage simulations of generated parametric jobs. Parametric simulations are often used for investigating the effects of a selected few design options, or exploring the solution space of all design options. Optimisation on the other hand, offers an efficient way to identify (near) optimum solutions within the whole solution space.

[3] Open Beagle: see http://beagle.gel.ulaval.ca/

[4] Matlab Global Optimisation Toolbox: see http://www.mathworks.co.uk/products/global-optimization/

[5] EnergyPlus: see http://apps1.eere.energy.gov/buildings/energyplus/

jEPlus stores definition of parameters of a building model, alongside the execution settings for EnergyPlus simulations, in a project file. The design parameters are organized in a tree structure, which has been described in detail in [6][7]. Figure 2 shows an example of parameter tree. Any part of a building model can be defined as a parameter, as long as it can be isolated within the model and be replaced with an alternative block by jEPlus. Each parameter definition contains a number of alternative values assigned by the user. For the example in Figure 2, there are four alternative building models for the building form parameter, whereas there are 24 alternative orientations for the north axis of the building. Note that the orientation of a building can be a continuous variable. It is discretized in the parameter definition.

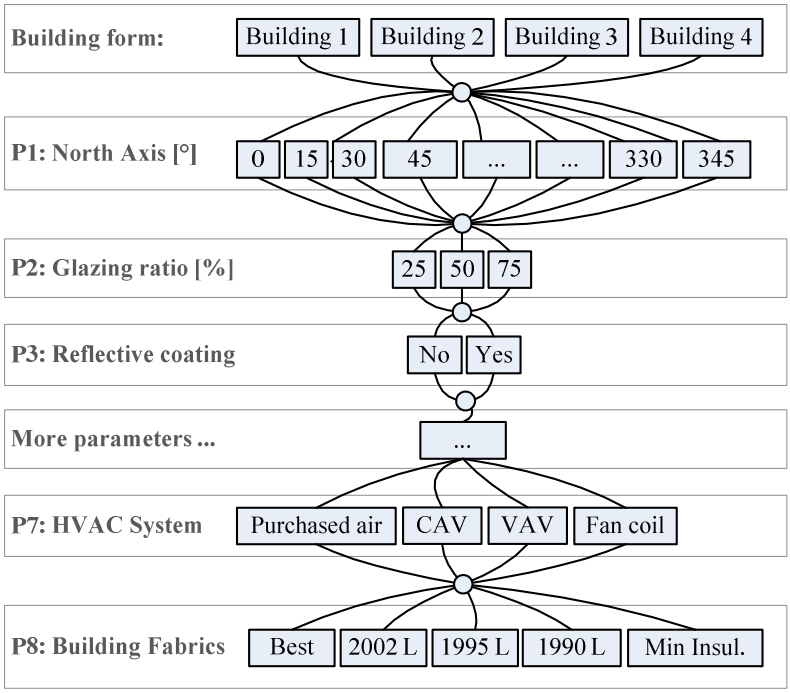

Figure 2 – jEPlus parameter tree

All design options that a user intends to investigate can be represented in a parameter tree. Each candidate solution contains all parameters, each with one alternative value assigned to it. Therefore, each path from top to bottom of the tree in Figure 2 represents one solution. If we put the index of the alternative value selected for each parameter together, we get an integer vector that can serve as an integer chromosome for EAs. For instance, a candidate with building form 2, rotation 45°, 25% of glazing with reflective coating, Fan coil system, and best practice insulation level can be represented as the vector (1 3 0 1 … 3 0)[6].

As we have discussed in section 4.0, the advantage of integer encoding is its efficiency. Binary encoding with fixed gene length is subject to bit redundancy, in which situation chromosomes may represent invalid or duplicate solutions. Real value encoding represents an infinite search space whenever continuous variables are involved. Effort has to be made to avoid GA searching in excessive resolution. Integer encoding, on the other hand, represents precisely the predefined solution space without any redundancy or duplication. In the case study in section 7, we provide a comparison of EAs performance between the jEPlus integer encoding and a binary encoding.

[6] Note that the indices are 0-based, i.e. the first element has an index 0, the second, 1, and so on.

6.0 Connecting EAs to jEPlus

Once the encoding scheme of the optimisation problem has been decided, and the solution space defined within a jEPlus project, an EAs tool can then be coupled with jEPlus to perform optimisation. All necessary input files for running EnergyPlus simulations are referenced in the jEPlus project file. These include the installation location of EnergyPlus executables, weather files, model files, the RVI/MVI file (instructions for result extraction), and the directory where simulation outputs will be stored. A user can use jEPlus' GUI to edit a project and test run a few simulations to make sure all settings are correct, before setting up optimisation.

List 1 shows a Java code sample with which jEPlus is called from the solution evaluation procedure in EAs (see Figure 1). The process contains loading a jEPlus project, creating a simulation manager with attached execution agent, converting chromosomes[7] in the current population to a jEPlus job set, running the jobs in the job set, and obtaining simulation results. The simulation results returned are double-precision vectors in a map indexed by job identifiers. The double-precision arrays holds any numerical information the user wants to extract (with a RVI/MVI file) from simulation outputs. These results can then be processed at the user's choice.

```
// … we are ready to run the simulations

// load project file
JEPlusProject Project = new JEPlusProject ( ”project_directory/project.jep”);

// create simulation manager
EPlusBatch SimManager = new EPlusBatch (null, Project);

// Set simulation agent
SimManager.setAgent(new EPlusAgentLocal ( Project.getExecSettings()));

// specify jobs; currentPopulation holds all chromosomes to be evaluated
String [][] jobs = new String [currentPopulation.getSize()][];
for (int i=0; i<currentPopulation.getSize(); i++) {
    jobs[i] = Project.getJobString(currentPopulation.getChromosome(i));
}

// execute jobs - job string contains values
SimManager.runJobSet(jobs);

// wait for jobs to finish
try {
    do {
       Thread.sleep(2000);
    }while (SimManager.isSimulationRunning());
}catch (InterruptedException iex) {
    SimManager.getAgent().setStopAgent(true);
}

// collect simulation results
HashMap Results = SimManager.getSimulationResults();

// … then process results using either single or multi-objective approaches
```

List 1 – jEPlus programming interface example

If a user's EAs tool does not support code-level Java interoperability, it can be

[7] In the current version (v1.2) of jEPlus at the time when this paper is prepared, JEPlusProject class supports both integer and binary chromosomes that comply with the JDEAL framework. JDEAL is the Java Distributed Evolutionary Algorithms Library: see http://www.laseeb.org/sw/jdeal/home.html

coupled with jEPlus using its command-line interface. In this case, user has to define the integer chromosome manually according to the jEPlus project, and then call jEPlus using the command shown in List 2.

```
Java –jar jEPlus.jar –job {project.jep} –index "{job string 1; job string 2; …}"
```

where "job string" is:

```
{job ID},{index of weather file},{index of building model},{index of parameter
    1}, {index of parameter 2}, …;
```

List 2 - jEPlus command-line interface

Text in '{ }' in List 2 must be replaced with the actual jEPlus project file name and job strings. Different job strings are separated with ';' character. A job string comprises a user-assigned "job ID" string for uniquely identifying the job, the index of the weather file, the index of the building model, and the index of each value selected for the corresponding parameters. On this command, jEPlus starts simulations immediately. When all jobs are completed, extracted results are collected into a text file, 'SimResults.csv', in the output directory specified in the project.

7.0 Case Study

We use an experiment to demonstrate performing optimisation using EAs coupled with jEPlus. Multi-objective and single objective approaches are tested on an optimisation problem. Also compared are binary encoding and jEPlus integer encoding schemes for their impact on optimisation performance.

The test case used in this experiment has been introduced in [7]. Four building forms included in the study are two typical office building shapes (square and rectangular) with two floor layouts (open-plan and cellular offices) each. Other design features and parameters considered as optimisation variables include orientation, glazing-to-wall ratio, glazing reflective coating, overhang size, daylight and glare control, selection of HVAC systems, and building fabrics. Details are listed in Table 1 below. In total, there are more than one million design solutions to explore. Optimisation objectives are to minimize annual heating and cooling demand of the building. Both single and multi-objective methods are tested.

Parameter	Values	Number of options
Building form (IDF model)	Rectangular-open plan, Rectangular-cellular offices Squar-open plan Squar-open plan + cellular offices	4
Orientation (North Axis - °)	0, 15, 30, 45, …, 345	24
Glazing ratio (%)	25, 50, 75	3
Glazing reflective coating ( )	Yes, No	2
Overhang depth (m)	0.2, 0.4, 0.6, …, 1.2	6
Daylight control ( )	None, daylight lighting control lighting control + glare control	3
HVAC system ( )	Ideal load system Variable air volume system Constant air volume system Fancoil + Fresh air system	4
Building fabrics ( )	Best practice, UK Building regulations 2002 Part L, UK Building regulations 1995 Part L, UK Building regulations 1990 Part L, Minimal insulation	5
Window construction ( )	Best practice, UK Building regulations 2002 Part L, UK Building regulations 1995 Part L, UK Building regulations 1990 Part L, Minimal insulation	5
Total options		1 036 800

Table 1 - Design variables

Table 2 shows the different EAs settings used for multi-objective and single objective optimisation in this case study. The NSGA-II algorithm [8] was used for multi-objective optimisation. Each experiment is limited to 100 generations. It was repeated 10 times. Unless otherwise stated, all results reported in this paper are average values from 10 separate trials. The experiments were carried out on De Montfort University's computer cluster using an extended version of jEPlus.

EAs setting	Integer encoding Multi-objective	Integer encoding Single objective	Binary encoding Single objective
Encoding	jEPlus integer (9 genes)	jEPlus integer (9 genes)	Bit string (9 x 10bit)
Algorithm	NSGA-II	Standard GA	Standard GA
Population size	20	20	20
Elistism size	Pareto optimal solutions	1	1
Selection	Binary tournament	Binary tournament	Binary tournament
Overall crossover rate	1.0	1.0	1.0
Uniform crossover option	25%	25%	50%
One-point crossover option	25%	25%	50%
Arithmetic crossover opt.	50%	50%	-
Overall mutation rate	0.1	0.1	0.1
Flip mutation option	50%	50%	100%
Gene/bit-wise mut. Rate	0.5	0.5	0.1
Shift mutation option	50%	50%	-
Gene/bit-wise mut. Rate	0.5	0.5	-

Table 2 - EAs settings

8.0 Results and analysis

Figure 3 shows the convergence chart of the optimisation process. Such charts are typically used for comparing performance of different EAs algorithms. On X-axis is the timeline (generation number). On Y-axis is the best objective function value found so far. The quicker and deeper a convergence line goes down, the better the performance of the algorithm. For the multi-objective trials, heating and cooling consumptions were separate objective functions for the optimisation process. A simple sum of heating and cooling consumptions is calculated and plotted on the chart after the trials have completed. For the single objective trials, the sum of heating and cooling consumptions was used as the objective function.

Integer encoding showed clear advantages over binary encoding, for the reasons explained in section 4.0. Although single objective approach converged faster at the early stage (till about the 15th generation) of optimisation, it tended to converge prematurely on a local optimum and therefore underperform in the long run compared to the multi-objective approach.

Premature convergence is a common problem with EAs. One of the strategies to tackle this is to perform several independent optimisation runs in parallel, therefore increase the chance of finding the global optimum. Table 3 compares the three methods in terms of the best solution found after certain number of simulations between 10 trials. The result further confirms that the single objective approach produces better results if the number of simulations is limited. On the other hand, multi-objective approach produces better results in the long run.

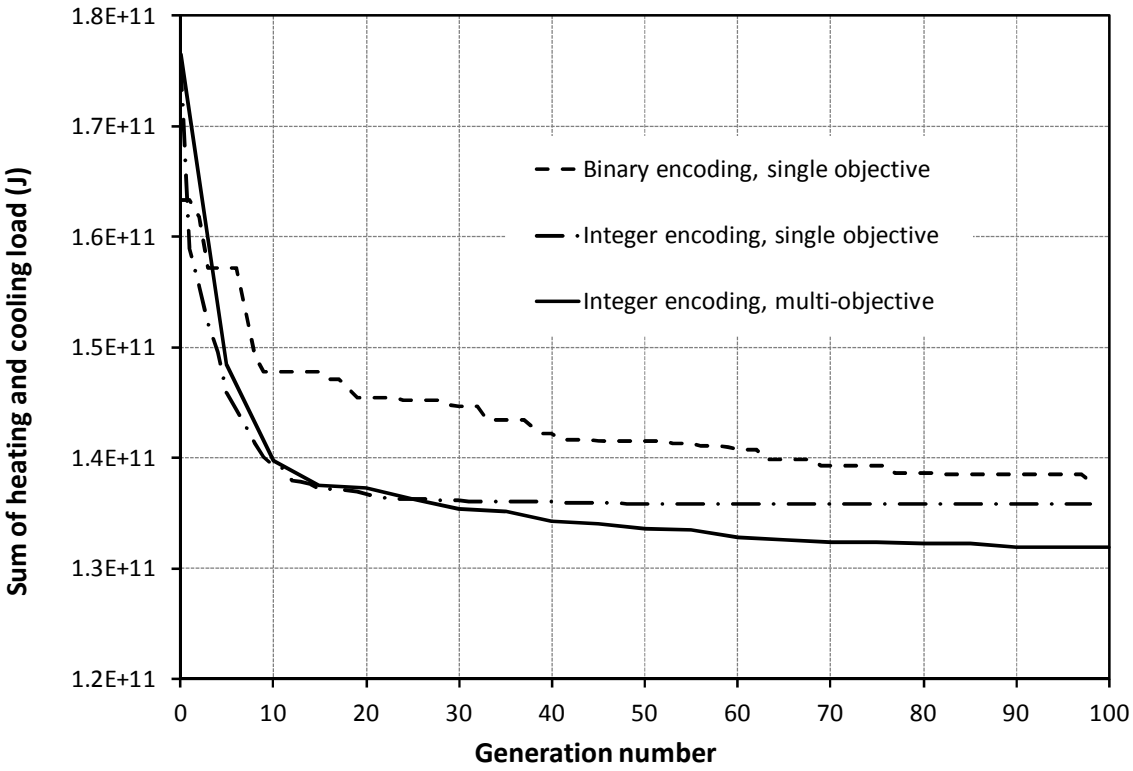

Figure 3 - Average convergence of objective function

Simulations performed	Integer Multi-objective	Integer Single objective	Binary Single objective
200	1.52994E+11	1.44950E+11	1.51717E+11
1 000	1.33839E+11	1.33839E+11	1.41224E+11
2 000	1.33839E+11	1.33650E+11	1.34115E+11
3 000	1.33839E+11	1.32730E+11	1.34115E+11
5 000	1.3365E+11	1.31919E+11	1.34115E+11
10 000	1.31411E+11	1.31411E+11	1.34115E+11
20 000	1.31336E+11	1.31411E+11	1.31461E+11

Table 3 - Best solution found in 10 parallel trials

Figures 4 and 5 explain why the multi-objective approach is less prone to premature convergence. Figure 4 shows the distribution of final solutions from the multi-objective approach. Only those considered Pareto optimal are plotting on the chart. It is clear to see these solutions were spread along the Pareto front. This helps keep divergence within the population; therefore, the algorithm is less likely to be trapped by a local optimum. In Figure 5, all solutions in the final population of the 10 trials are plotted. The solutions were tightly clustered around the (local) minima, which hindered further exploration of better solutions.

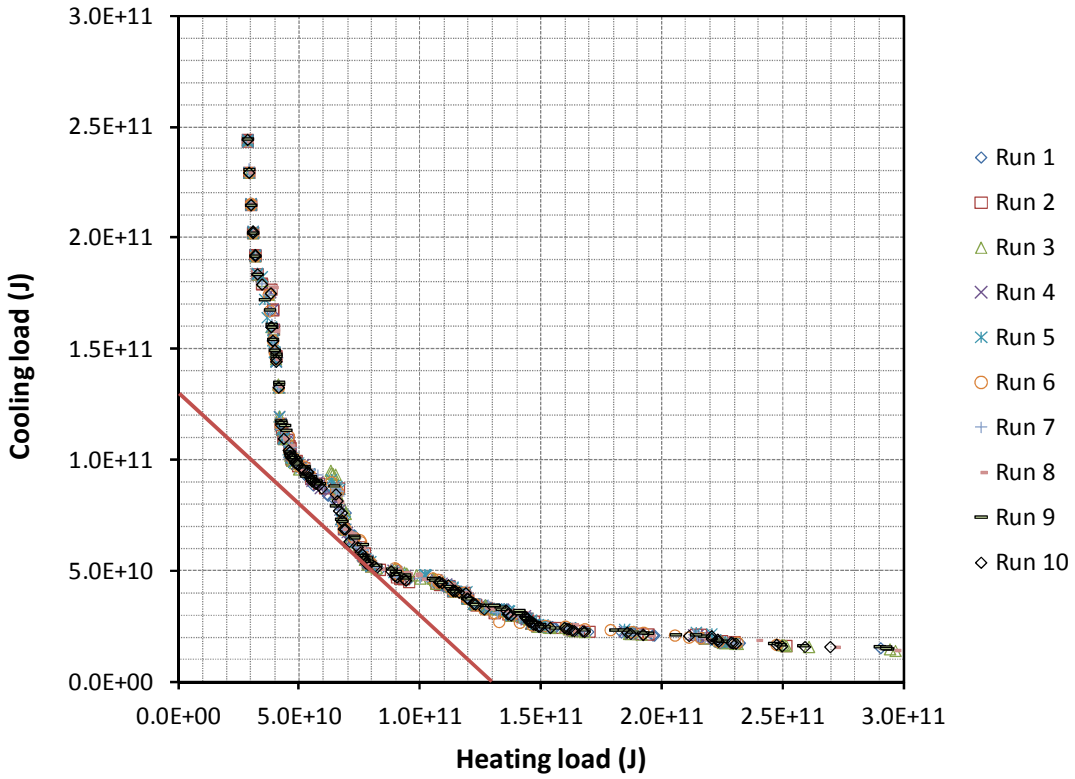

Figure 4 – Distribution of pareto optimal solutions in the final population of all multi-objective trials

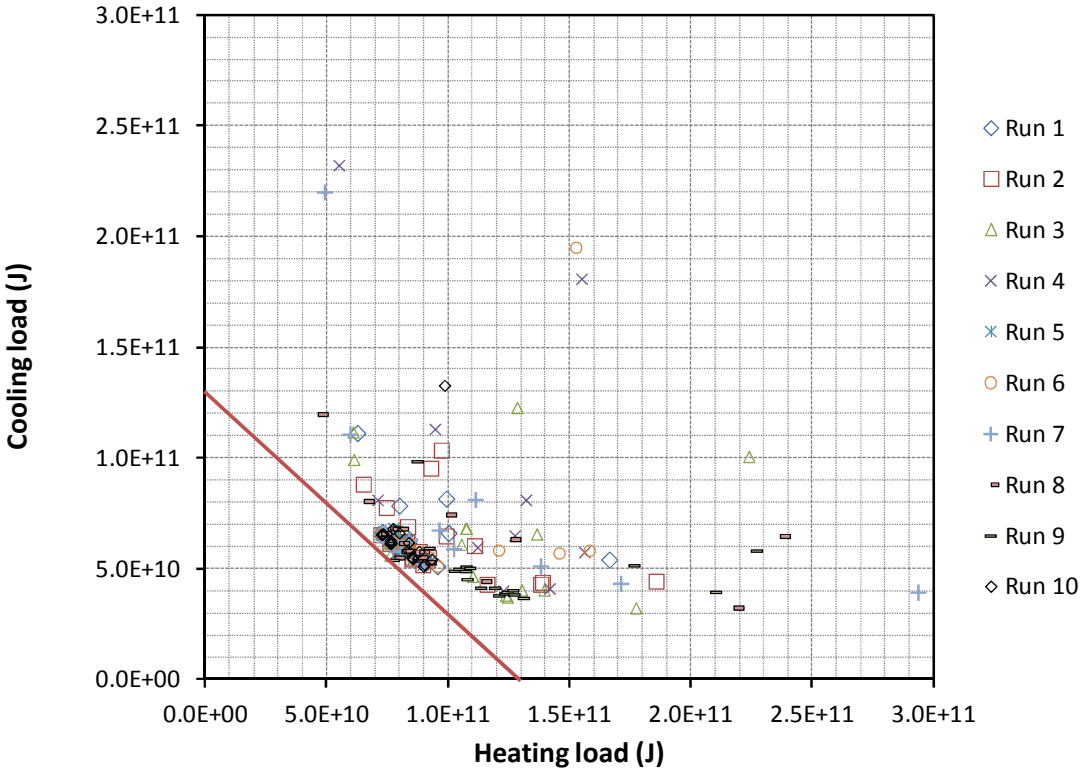

Figure 5 – Distribution of all solutions in the final population of single objective (integer encoding) trials

9.0 Conclusions

jEPlus provides a convenient way to interface EnergyPlus models with optimisation algorithms. It simplifies three key tasks: encoding of a building design problem, mapping a design solution to a simulation model, and execution of simulation jobs. Users can use either the programming interface or the command-line interface to couple their own optimisation algorithms to jEPlus.

Evolutionary Algorithms (EAs) is one of the most widely used optimisation methods in engineering fields. A case study of the application of EAs in building design is presented in the paper. We demonstrated the use of both multi-objective and single objective approaches on minimising heating and cooling consumptions of an office building. Both methods showed quick convergence of the objective function(s). The single objective approach, in which the simple sum of the heating and cooling consumptions is used as the objective function, converged quicker but was prone to being trapped in a local optimum. The multi-objective approach produced better solutions consistently in the experiments. It was also shown that the jEPlus integer encoding scheme has significant performance advantage over a typical binary encoding scheme.

References

[1] Wetter, M. 2001. GenOpt - A generic optimization program. Proc. of the 7th IBPSA Conference, volume I, pages 601-608. Rio de Janeiro, 2001.

[2] Ellis, P. G. Griffith, B. T. Long, N. Torcellini, P. and Crawley, D. 2006. Automated multivariate optimization tool for energy analysis. Proceedings of the SimBuild 2006 Conference, August 2-4, 2006, Cambridge, MA.

[3] Wright, J. A., Zhang, Y., Angelov, P.P., Buswell, R. A., Hanby, V. I. 2008. Evolutionary Synthesis of HVAC System Configurations: Algorithm Development (RP1049), HVAC&R Research, Vol. 14(1):33-55

[4] Palonen, M., Hasan, A. and Siren, K. 2009. A genetic algorithm for optimization of building envelope and HVAC system parameters. Proc. of the 11-th IBPSA Conference, Glasgow, UK, July 2009.

[5] Coffey, B. Haghighat, F. Morofsky, E. and Kutrowski, E. 2010. A software framework for model predictive control with GenOpt. Energy and Buildings, Volume 42, Issue 7, July 2010, Pages 1084-1092

[6] Zhang, Y. 2009. 'Parallel' EnergyPlus and the development of a parametric analysis tool, Proc. of the 11th IBPSA Conference, Glasgow, UK, July 2009.

[7] Zhang, Y., Korolija, I. 2010. Performing complex parametric simulations with jEPlus, SET2010 - 9th International Conference on Sustainable Energy Technologies, 24-27 August 2010, Shanghai, China

[8] Deb, K., Pratap. A, Agarwal, S., and Meyarivan, T. 2002. A fast and elitist multi-objective genetic algorithm: NSGA-II. IEEE Transaction on Evolutionary Computation, 6(2), 181-197.