Building Simulation 2003

Eighth International IBPSA Conference
Eindhoven, Netherlands
August 11-14, 2003

INTEGRATION OF AN INTERNAL OPTIMIZATION MODULE WITHIN ENERGYPLUS

G. Zhou[1], P. Ihm[1], M. Krarti[1], S. Liu[2] and G.P. Henze[2]

[1]Civil, Environmental, and Architectural Engineering, University of Colorado
[2]Architectural Engineering, University of Nebraska-Lincoln

ABSTRACT

An optimization module is developed and incorporated within *EnergyPlus*. As an application of the optimization module, improved controls are determined for building passive thermal energy storage inventory. The paper describes the implementation of the optimization module within *EnergyPlus*. Moreover, the paper presents results of a comparative analysis to assess the performance of various optimization algorithms evaluated as part of the implementation of the internal optimization module within *EnergyPlus*. Parametric analyses are carried out to evaluate the effectiveness of the optimization module integrated with *EnergyPlus* in harnessing the building thermal mass to reduce either energy use or peak-demand associated with cooling a prototypical office building under various operating conditions. Selected results of the parametric analyses are presented in this paper. In particular, the performance of the internal optimization module is compared against that of *GenOpt*, an external optimization system. The study reveals that the Nelder-Mead simplex method provides an appropriate compromise between computational efficiency, robustness, and accuracy of the optimal solution.

INTRODUCTION

EnergyPlus is a new generation of building energy simulation software developed by the U.S. Department of Energy (DOE, 2002). The first version 1.0.0 was released in April 2001 and the latest version 1.0.3 was released in November 2002. The next major release is scheduled to be released March 2003.

EnergyPlus is built on popular features and capabilities of both BLAST and DOE-2 but also includes several innovative simulation capabilities such as flexible sub-hourly time steps, modular HVAC systems, multi-zone airflow, thermal comfort, and photovoltaic systems. *EnergyPlus* is expected to be a valuable tool to simulate building energy use and study control strategies of building mechanical systems to reduce both building energy consumption and operating costs.

Like other whole-building simulation programs, *EnergyPlus* simulates building energy flows based on an input file containing a detailed description of building construction, HVAC systems and their controls. Typically, the user defines building operation and control strategies based on experience or existing operating schemes. Often, these user-defined controls are ad hoc and hence do not provide optimal operating strategies, resulting in higher than necessary energy use and/or energy cost. To broaden the applications of *EnergyPlus*, an internal optimization module is developed and integrated so that *EnergyPlus* can be used to find an optimal control strategy for operating HVAC systems.

In particular, this paper describes and evaluates an approach to implement suitable optimization routines within *EnergyPlus* in order to cost-effectively utilize passive building thermal energy storage. However, the developed optimization module could be applied to better control any energy system within a building.

OVERVIEW OF ENERGYPLUS

In order to better understand the integration of optimization routines, it is valuable to outline the basic structure of *EnergyPlus* to calculate building cooling and heating loads and simulate the operation of HVAC systems and central plant equipment. Refer to DOE (2002) for detailed information on *EnergyPlus*. Refer to Appendix A for description of the basic structure and building control strategies of *EnergyPlus*.

OPTIMIZATION ALGORITHMS

Numerous optimization techniques can be considered for the implementation within *EnergyPlus* to carry out cost or energy minimization. Each technique has its advantages and disadvantages depending on the building system to be controlled. In particular, only selected optimization techniques are suitable for controlling building passive or active thermal energy

storage (TES) systems due to several constraints including:

1. The interactions between TES systems and other building HVAC systems are rather complex. In particular, no simple and differentiable relationship can be generally established between TES control variables (such as indoor air temperature setpoints or charge/discharge levels) and building energy consumption. Therefore, the optimization objective function is not generally differentiable. Given this characteristic, optimization techniques that require as input parameters the derivatives of the objective function are not suitable for TES optimization problems.
2. Acceptable control variables for the optimization of TES systems are within simple bounds such as the upper/lower limits of storage levels for active TES systems and upper/lower limits of zone air temperatures for passive TES systems. Therefore, suitable optimization algorithms have to account for bound constraints in their search. Unconstrained optimization approaches are therefore not appropriate for TES optimization problems.
3. The input/output formatting and the structure of the optimization algorithms should be flexible enough to be compatible with *EnergyPlus*. Most of the commercially available optimization programs cannot be easily implemented within *EnergyPlus* because of their rigid structure. It should be noted that commercially available optimization software could be interfaced with *EnergyPlus* using an external optimization approach such as *GenOpt*.
4. Given these constrains and the accessibility of source codes, the optimization algorithms considered for this study are based on the following methods:
   - Nelder-Mead Simplex
   - Quasi-Newton as proposed by Broyden-Fletcher-Golfarb-Shanno (*BFGS*)
   - Simulated Annealing
   - Population-Based Approach

A brief discussion of each optimization method as well as their associated algorithms is provided in Appendix B.

IMPLEMENTAION OF AN OPTIMIZATION ALGORITHIN WITHIN ENERGYPLUS v.1.01

Optimization Algorithm

As discussed in Section 3, several criteria should be considered to select a suitable optimization algorithm for implementation within *EnergyPlus*. For this study, four optimization algorithms are implemented and evaluated. The implementation procedure for all the optimization algorithms is similar. To illustrate the implementation procedure within *EnergyPlus*, an algorithm using the Nelder-Mead simplex method is considered in this section. The algorithm is extracted from the ISML library and is called *DBCPOL* program.

ManageSimulation subroutine

As described in Section 3, HVAC systems in *EnergyPlus* can be controlled by utilizing schedules. Some modifications of internal subroutines are required in order to implement an optimization algorithm within *EnergyPlus* as discussed below.

First, subroutine *ManageSimulation* is modified to separate the initialization tasks from the iterative calculations. This separation is simply achieved by code after the line "*BeginFullSimFlag = True*", indicating the completion of initialization, from the subroutine *ManageSimulation* to another subroutine *EnvironSimulation* where the environment loop simulation starts.

Also, in order to keep the flexibility of running EnergyPlus with and without the optimization module, a new do-optimization-or-not field is added into section Run Control in the input file to allow the user to select the simulation type. If the input value for the do-optimization-or-not field is 'YES', then the newly created flag *DoOptimization* is set to "true" and EnergyPlus with the integrated optimization algorithm is used for the simulation. Otherwise, standard EnergyPlus simulation is executed without any optimization.

In subroutine ManageSimulation, instead of starting the environment simulation directly by calling *EnvironSimulation*, the simulation can be directed into two newly added subroutines: *ManageNormalEnergyPlus* or *ManageStdOptimEnergyPlus* depending on the value of the flag *DoOptimization*. If *DoOptimization* is 'false', *ManageNormalEnergyPlus* is initiated by calling directly *EnvironSimulation*. Otherwise, *ManageStdOptimEnergyPlus* is initiated by first calling another initialization subroutine, *ProcessOptimizationInput*, to obtain input values

needed by the optimization subroutine, *BCPOL*. Then, the subroutine *BCPOL* is called within *EnergyPlus* to start the optimization process. In this process, the *EnvironSimulation* subroutine determines the objective function. Using the *BCPOL* algorithm, several iterations are typically required to determine the optimal values of the control variables. For each iteration, *BCPOL* updates the values of the control variables and recalculates the objective function by calling the *EnvironSimulation* subroutine (indirectly). After *BCPOL* finds the optimal control variable values, *EnvironSimulation* subroutine is called one more time and the output values are reported.

Input Parameters for the Optimization subroutine

The *BCPOL* algorithm requires eight input variables and provides two output variables as described in Section 4.1. These input and output variables are defined in the module *DataOptimization*, a global data module that can be accessed by other modules. The defined variables are used by *DBCPOL* routine as follows:

```
CALL DBCPOL (FCN, NumVariables, Guess,
IBTYPE, LowerLimit, UpperLimit, Tolerance,
MaxRunningNum, OptimumX, OptimumValue)
```

The values for the input variables are obtained by calling subroutine *ProcessOptimizationInput* in file *ScheduleManager*. In order for the BCPOL to know which control variables need to be optimized, a new section named *Optimization:Schedule* is created in the input file, which has the format shown below.

```
OPTIMIZATION: SCHEDULE,
  A1,     \field OptimizationScheduleName
          \Required-field
          \Type alpha
  A2,     \field Guess Schedule
          \Required-field
          \Type object-list
          \Object-list ScheduleNames
  A3,     \field Lower Limit Schedule
          \Required-field
          \Type object-list
          \Object-list ScheduleNames
  A4;     \field Upper Limit Schedule
          \Required-field
          \Type object-list
          \Object-list ScheduleNames
```

Each control variable to be optimized should have a corresponding *Optimization:Schedule* section in the input file. Field A1 is the name of the control variable schedule defined in the input file. Field A2 is the name of the schedule that provides guess values of the control variables. Fields A3 and A4 are the names of the schedules that provide the lower and upper bounds of the control variables, respectively. The construction of this section corresponds to the *OptimizationScheduleData* data type defined in a newly added global data file *DataOptimization*. In the newly added *ProcessOptimizationInput* subroutine, the schedule indices of these schedules in the *Optimization:Schedule* sections are stored in the array *OptimizationSchedule* whose format is defined as data type *OptimizationScheduleData*. Since *EnergyPlus* already features routines that can retrieve the schedule value by the schedule index, it is straightforward to obtain values for all schedules defined in the *OptimizationSchedule* array.

Hence, once the schedule names of the control variables are read into *OptimizationSchedule*, it is easy to get their values and store them in arrays that serve as input for the BCPOL algorithm by using subroutines such as *GetScheduleIndex* and *GetCurrentScheduleValue*. Therefore, in subroutine ProcessOptimizationInput, arrays for *Guess*, *LowerLimit*, *UpperLimit*, *OptimumX* are allocated according to the number of control variables that need to be optimized, and the schedule values of these control variables are stored in *Guess, LowerLimit*, and *UpperLimit* sequentially. The remaining input parameters for the BCPOL algorithm include *NumOfVariables, Tolerance, IBTYPE, MaxRunningNum* are also assigned values by calling *ProcessOptimizationInput*.

Finally, other subroutines are added or modified within *EnergyPlus* to complete the implementation of an internal optimization module including:

- *BCPOL* algorithm calls subroutine *FCN* to be called by *BCPOL* to calculate the objective function. This subroutine is located in the *SimulationManagerFile*.
- An economical module calculates the total electrical utility costs including demand and energy charges. The economical is located in the *EnvironSimulation.*
- A new section is added to allow more flexibility in the selection of the objective function; the user can choose between energy cost only, demand cost only, or total cost. This section is processed in *ProcessOptimizationInput* and the value is stored

in a variable called *OptimizationCost* defined in *DataOptimization* as shown below.

```
OPTIMIZATION:COST,
    N1;     \field Optimization Cost
            \Type choice
             \Key 1 !–energy cost based
             \Key 2 !–demand cost based
             \Key 3 !–total cost based
```

Output Variables

With standard *EnergyPlus*, the environment simulation is run only once. However, *EnergyPlus* with an internal optimization module requires several simulation iterations to find the optimal solution. Without changing reporting features, *EnergyPlus* would provide lengthy output reports for all the simulation iterations. To eliminate any intermediate and undesirable output reports, a variable *EndOptimizationFlag* is defined In *DataOptimization*. This variable is set to "false" until the optimization is done. Only before the final simulation run, *EndOptimizationFlag* is set to "true". The output reporting for the standard *EnergyPlus* (i.e., without the optimization module) is controlled solely by *DoOutputReporting* variable. In the modified *EnergyPlus* (i.e., with the optimization module), both *DoOutputReporting* and *EndOptimizationFlag* variables control the output reporting.

Thermal History

In order for algorithm BCPOL to efficiently call subroutine *EnvironSimulation* to perform several simulations, other modifications are introduced in *EnergyPlus* related to the counting of environments and to tracking the building's thermal history:

EnvironSimulation uses subroutine *GetNextEnvironment* to assess whether or not there is another environment available for simulation. The number of environments is counted from the input file during initialization of *EnergyPlus*, i.e., before calling *EnvironSimulation* using an internal counter. This internal counter increases by one every time *GetNextEnvironment* subroutine is called. In order for BCPOL algorithm to call *EnvironSimulation* as many times as needed, the environment counter in *GetNextEnvironment* subroutine is reset after each *EnvironSimulation* iteration.

When calling *EnvironSimulation* before the beginning of any simulation, *EnergyPlus* executes a "warm-up" period to initialize and determine the actual thermal history for the building thermal zones and the HVAC systems. This warm-up period is determined starting from an arbitrary thermal history. Then, by repeatedly running the first day of that environment until the thermal history related variables converge, the warm-up period is completed and the thermal history at the start of the first timestep of that environment is found. Then, based on this initial thermal history, the simulation starts. Without modifying the warm-up period input, the warm-up period always uses the same input as the first day of that environment. With an internal optimization module, this thermal history setting in *EnergyPlus* is incorrect and has to be modified. Indeed, the *BCPOL* algorithm changes values for various control variables for a given environment during each iteration. As a consequence, the warm-up period and thermal history change according to the control variable values set at each iteration. However, the actual thermal history at the beginning of the environment under investigation should not be changed using future operation strategies but should only depend on past operation conditions. Thus, instead of using the characteristics of the "first day" of an environment as input variables for the warm-up period, the characteristics of "yesterday" should be used instead. After the warm-up period is complete, the thermal history at the start of the first timestep of the environment should be recorded and should not be changed later. The calculation of the objective function values in each iteration of *BCPOL* should be based on this same starting thermal history.

To be able to record the thermal history at the end of the warm-up period and upload the thermal history at the start of the actual simulation, variables that store building and systems related thermal histories need to be identified and subroutines that record and update thermal history values are needed. Since in the released *EnergyPlus* documents it is not revealed which variables store thermal history, the identification of these thermal history related variables was conducted by trial-and-error. File *ThermalHistoryManager* is added to *EnergyPlus*, which includes public subroutines that record and update thermal histories, i.e., *RecHistoryValues* and *UpdateHistoryValues*.

Since in *EnergyPlus* there is already a logical variable, *WarmUpFlag*, marking the beginning and the end of the warm-up period, it is easy to record the thermal history after the warm-up period is completed and upload recorded thermal history at the start of the first timestep. Code recording and updating thermal histories are added in the timestep loop of *EnvrionSimulation*.

To ensure that the warm-up process is based on "yesterday's" weather data, subroutines in *WeatherManager* are modified. Unlike other input variables in *EnergyPlus*, weather data are read day-by-

day instead of timestep-by-timestep. Before the beginning of each day, the weather data are read into a global variable *Tomorrow* by calling *ReadWeatherForDay* subroutine. At the beginning of that day, global variable *Today* is updated by Tomorrow by calling *UpdateWeatherForDay* subroutine. This process is accomplished in subroutine *InitializeWeather* in *WeatherManager*. Both *Today* and *Tomorrow* variables store weather data such as outside dry bulb/dew point temperatures, pressure, solar radiation, wind speed and direction. For standard *EnergyPlus*, the warm-up period reads the weather information of the first day of the environment. *EnergyPlus* with an internal optimization module reads instead weather data for the day before the first day of the environment. Also, the date is changed to one day before the first day so that the building schedules are those of yesterday.

DISCUSSION SELECTED RESULTS

In this section, selected simulation results obtained from *EnergyPlus* with an internal optimization module are presented. The main focus of the analysis is to determine the best control strategies that utilize passive building thermal energy storage in a typical office building in order to reduce electrical energy charges, electrical demand charges, or total electricity charges. First, a brief outline of the building model used throughout the simulation analysis is provided. Then, sample of optimization results for passive TES system is illustrated. Finally, a comparative analysis of the performance of optimization algorithms is presented.

Building Model

A 3-floor office building model is used to evaluate the performance of various optimization algorithms within *EnergyPlus*. Figure 1 shows an isometric view of the office building.

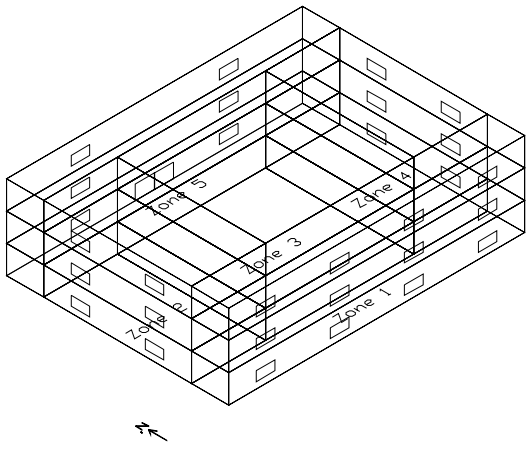

Figure 1: Isometric view of the office building model

Each floor of the building is modeled with five zones, i.e. east (24m x 12m), south (48m x 6m), west (24m x 12m), north (48m x 6m) and central zone (24m x 24 m). The total building floor area is 5,184 m^2. Two construction types are considered in the analysis: heavy-mass (643 kg/m^2) and light-mass (183 kg/m^2). The building is occupied 8:00-19:00 and unoccupied for the rest of the day. The average people density is set at 0.08 $person/m^2$ for all zones. The density load attributed to both lighting and equipment is 31.25 W/m^2. Two electrical utility rates, i.e. weak rates and strong rates, are considered in the analysis. In weak rates, the on-peak and off-peak energy costs are 0.2 \$/KWH and 0.1 \$/KWH respectively and demand costs are 10 \$/KW and 5\$/KW respectively. In heavy rates, the energy costs are 0.2 \$/KWH and 0.05 \$/KWH and demand costs are 10 \$/KW and zero \$/KW respectively. The simulation runs are performed either for Jan 17th or for July 20th using weather data of Phoenix, AZ.

Sample of *EnergyPlus* based Optimization Controls

Figure 2 shows sample results, obtained by *EnergyPlus* with an internal optimization module, to better control zone cooling setpoints in order to reduce total electricity charges (using the strong-incentive utility rate structure) for the 15-zone office building during July 21 in Phoenix, AZ. *EnergyPlus* predictions for zone cooling setpoints, average zone air temperatures, as well as the chiller power consumption are illustrated in Figure 3 for both conventional control (i.e., night set-up control) and optimal control (using pre-cooling strategies). In the simulation, a temperature economizer is assumed for the operation of the HVAC systems.

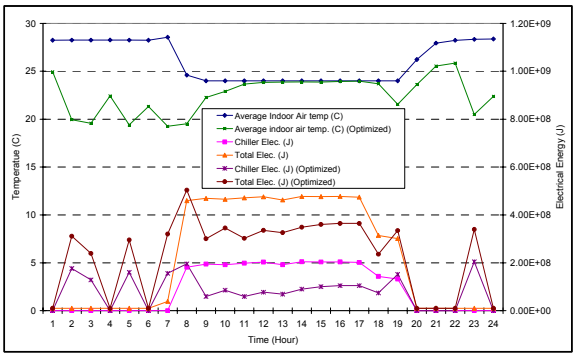

Figure 2: Optimal control of passive building energy thermal storage for the15-zone office building using EnergyPlus

From Figure 2, it can be observed that the optimal solution decides to precool the building during nighttime. During the day, the cooled building structure absorbs zone internal gains resulting in reduced cooling load and chiller power consumption compared to the

conventional control. For the optimal control, the total chiller power consumption is 554 kWh with a peak of 56.7 kW occurring at 23:00. The building total electrical consumption is 1582 kWh with a peak of 139.9 kW occurring at 8:00. Compared to the conventional control, applying optimal control can achieve total cost savings of 19.6% and a cooling cost savings of 33.0%.

Comparative analysis of the optimization algorithms

The performance of four optimization algorithms, i.e. BCPOL, BCONF, SIMANL, and OptQuest is compared for various operating and design conditions including the effects of building mass, electrical utility rate, and weather conditions. Table 1 lists selected the results of the comparative analysis. Figure 3 illustrates typical performance of the four optimization algorithms.

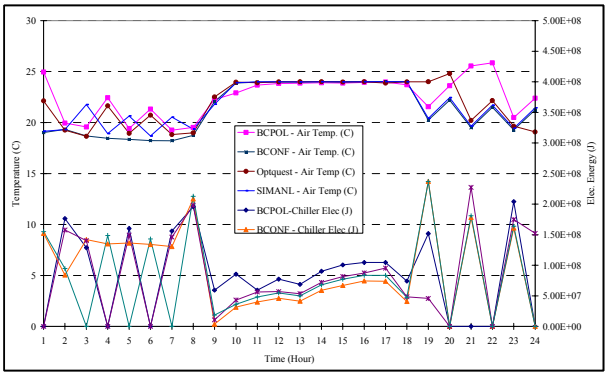

Figure 3: Temperature and electricity consumption profiles obtained using four optimization algorithms integrated within EnergyPlus to better utilize building thermal mass

From Figure 3, it can be observed that all of the optimization algorithms provide rather non-smooth solutions as indicated by the ragged profiles of the zone air temperature and chiller energy use. It is most likely that most of the optimization algorithms find local optima instead of the global optimum due to the complexity of the optimization problem.

As indicated in Table 1, SIMANL algorithm requires a significant CPU time to search for the optimal solution. Indeed, to find the optimal temperature setpoints, SIMANL needed more than 50 hours and 31201 function evaluations. Although SIMANL achieved the highest savings, the long simulation time makes it unsuitable for most optimization problems.

Similarly, the OptQuest algorithm requires long CPU times. In order to get acceptable optimal solutions, approximately 5,000 iterations needed. The average CPU time OptQuest required in this scenario is about 6 hours. Comparing the total cost savings summarized in Table 1 and obtained by various optimization algorithms, it can be observed that OptQuest achieves 0.5-7% higher savings than BCPOL and 0.7%-11.4% more savings than BCONF. Although it may provide better optimal solutions, OptQuest is not suitable for real-time optimal control problems due to the large computational efforts required.

Finally, the performance of BCPOL and BCONF algorithms is comparable and both require similar CPU times and iterations to find optimal solutions. On average, BCPOL needs 1,053 of iterations and 1 hour 27 minutes of CPU time while BCONF requires 857 iterations and 57 minutes of CPU time. The differences for the optimal objective function values between the two algorithms are rather small (in the range 0.1-7%). However, BCONF is not as robust as BCPOL due to the fact gradients for the objective function are needed by the quasi-Newton technique that BCONF utilizes to search for the optimal solutions. Considering both robustness and computational efficiency, it appears that the BCPOL algorithm represents the best choice among the four optimization algorithms considered in the comparative analysis.

Comparison with an external optimization module, *GenOpt*

GenOpt® is a generic optimization program developed by the Simulation Research Group at the Lawrence Berkeley National Laboratory. It can be applied to optimize any arbitrary objective function that can be estimated by an external simulation program such as *EnergyPlus*. There are currently two optimization methods integrated within *GenOpt* that the user can select from. For fair comparison, the performance of the Nelder-Mead simplex method of *GenOpt* is evaluated against the BCPOL algorithm integrated within *EnergyPlus*. Table 2 summarizes the comparative analysis of the performance of both GenOpt (externally connected to EnergyPlus) and BCPOL (internally integrated within EnergyPlus) in determining the best control strategy for passive building storage utilization to minimize total electricity costs for the 15-zone office building located in Phoenix AZ, during July 21. Two electrical utility rate structures (weak-incentive vs. strong-incentive) and tow construction types (heavy vs. light) are considered in the comparative analysis.

From Table 2, it can be noted that the Simplex method of GenOpt tends to get easily trapped in local minima. For the weak-incentive rate, it cannot find a solution that is better than the base case, which is night set-back control. This is due to the intentional reduction in the number of iterations that the Nelder-Mead method is

allowed to perform in *GenOpt* to search for the optimal solution. This limit on the number of iterations is set to reduce the CPU time for the search.

Further, it can be observed that although the Simplex method in GenOpt requires fewer function evaluations to get the "best solution", it takes significantly longer in terms of CPU time (on average 6 hours). This low computational efficiency is due to two facts. First, communication between *GenOpt* (an external optimization module) and *EnergyPlus* occurs through file transfer that requires more CPU time than that needed by an optimization module integrated within *EnergyPlus*. Second, due to the structure of *GenOpt*, EnergyPlus is called at every iteration (of the search process) with subsequent CPU time-consuming of warm-up period and initialization of building energy systems.

Based on the comparative analysis results, it is clear that an internal optimization module performs better than an external optimization module in cases when high computational efficiency is required such as real-time optimal control.

SUMMARY AND CONCLUSIONS

A description of the implementation of an optimization algorithm within *EnergyPlus* is presented in this paper. Several parametric analyses have been carried out using *EnergyPlus* with an internal optimization module to determine the best control strategies for passive building thermal storage inventory in buildings. The impact of the optimization technique is evaluated under various operating and design conditions. Four optimization techniques are considered in the evaluation analysis including:

- Nelder-Mead Simplex
- Quasi-Newton as proposed by Broyden-Fletcher-Golfarb-Shanno (BFGS)
- Simulated Annealing
- Population-Based Approach

Considering both robustness and computational efficiency of the optimization routines, the BCPOL algorithm using the Nelder-Mead simplex method was found to be the best choice for the specific problem of optimizing building passive TES systems. Comparing external and internal implementations for the optimization module, it was found that the internal implementation provides significantly more efficient option.

ACKNOWLEDGMENT

The authors gratefully acknowledge the support of this work through U.S. Department of Energy Cooperative Agreement No. DE-FC26-01NT41255.

REFERENCES

Belegundu, A.D., Chandrupatla, T.R., 1999. Optimization Concepts and Applications in Engineering, Upper Saddle River, N.J., Prentice Hall.

Broyden, G., 1970. The Convergence of a Class of Double-Rank Minimization Algorithms, Parts I and II, Journal of the Institute of Mathematics and Its Applications, Vol. 6, 76-90 and 222-231.

DOE, 2002. EnergyPlus Documentations and Manuals, US Department of Energy, also refer to http://www.eere.energy.gov/buildings/energy_tools/energyplus/.

Fletcher, R., 1970. A New Approach to Variable Metric Algorithms, Computer Journal, Vol. 13, 317-322.

Glover, F., Laguna, M., 2000. Tabu Search, published by Kluwer Academic Publishers.

Goldfrab, D., 1970, A Family of Variable Metric Methods Derived by Variations Means, Mathematics of Computation, Vol. 24, 23-26.

Nelder, J.A., Mead, R., 1965. A Simplex Method for Function Minimization, Computer Journal, Vol. 7, 308.

Rao, S.S., 1996. Engineering Optimization: Theory and Practice, Third Edition, John Wiley and Sons Inc.

Shanno, D.F., 1970. Conditioning of Quasi-Newton Methods for Function Minimization, Mathematics of Computation, Vol. 24, 647-656.

http://www.vni.com/products/imsl/docs/MATH.pdf, page 958

http://www.vni.com/products/imsl/docs/MATH.pdf, page 933

http://cxc.harvard.edu/ciao/ahelp/simul-ann-1.html

http://www.opttek.com/products/optquest_eng.html

http://simulationresearch.lbl.gov/GO/

http://simulationresearch.lbl.gov/GO/download/documentation.pdf, page 25

Due to space constrains, appendices A and B are only available in the electronic version of the paper.

Table 1

Comparison of the performance of four optimization algorithms integrated within *EnergyPlus*.

		NO Opt.	BCPOL				BCONF				OptQuest				SIMUAL			
		Costs	Costs	Savings	# Iter.	Time	Costs	Savings	# Iter.	Time	Costs	Savings	# Iter.	Time	Costs	Savings	# Iter.	Time
Heavy-	Total Cost ($)	310.31	249.62	-19.6%			239.66	-22.8%			241.92	-22.0%			235.63	-24.1%		
Strong-	Cooling Cost ($)	183.64	122.95	-33.0%	1317	2:12	112.98	-38.5%	1746	1:40	115.25	-37.2%	5000	8:31	108.96	-40.7%	31201	51:02
Summer	Chiller Elec. Cost ($)	130.81	71.73	-45.2%			57.01	-56.4%			61.77	-52.8%			57.78	-55.8%		
Heavy-	Total Cost ($)	369.38	378.23	2.4%			380.99	3.1%			352.77	-4.5%						
Weak-	Cooling Cost ($)	215.48	232.89	8.1%	927	1:50	235.64	9.4%	1112	1:46	207.42	-3.7%	5000	6:00				
Summer	Chiller Elec. Cost ($)	152.29	147.22	-3.3%			145.18	-4.7%			134.65	-11.6%						
Light-	Total Cost ($)	317.43	268.49	-15.4%			265.20	-16.5%			262.37	-17.3%						
Strong-	Cooling Cost ($)	190.75	141.82	-25.7%	1214	1:47	138.52	-27.4%	568	0:34	135.69	-28.9%	5000	6:40				
Summer	Chiller Elec. Cost ($)	136.63	88.00	-35.6%			79.69	-41.7%			82.08	-39.9%						
Light-	Total Cost ($)	376.24	364.46	-3.1%			366.11	-2.7%			359.93	-4.3%						
Weak-	Cooling Cost ($)	222.34	225.94	1.6%	834	1:15	229.47	3.2%	1156	1:44	214.58	-3.5%	5000	4:58				
Summer	Chiller Elec. Cost ($)	157.82	143.39	-9.1%			146.65	-7.1%			142.20	-9.9%						
Heavy-	Total Cost ($)	164.31	139.06	-15.4%			154.45	-6.0%			135.68	-17.4%						
Strong-	Cooling Cost ($)	37.64	12.39	-67.1%	954	1:24	27.77	-26.2%	417	0:24	9.00	-76.1%	5000	5:49				
Winter	Chiller Elec. Cost ($)	2.97	0.13	-95.6%			2.51	-15.5%			0.00	-100.0%						
Heavy-	Total Cost ($)	196.59	181.11	-7.9%			190.75	-3.0%			178.45	-9.2%						
Weak-	Cooling Cost ($)	43.92	28.95	-34.1%	1054	0:49	38.16	-13.1%	648	0:34	28.57	-34.9%	5000	4:58				
Winter	Chiller Elec. Cost ($)	2.97	3.21	8.1%			2.65	-10.8%			0.00	-100.0%						
Light-	Total Cost ($)	166.47	140.35	-15.7%			144.23	-13.4%			139.63	-16.1%						
Strong-	Cooling Cost ($)	39.79	13.67	-65.6%	1208	1:31	17.56	-55.9%	805	0:42	12.95	-67.5%	5000	6:23				
Winter	Chiller Elec. Cost ($)	4.40	0.02	-99.5%			0.31	-93.0%			0.00	-100.0%						
Light-	Total Cost ($)	196.23	181.59	-7.5%			196.40	0.1%			180.17	-8.2%						
Weak-	Cooling Cost ($)	44.90	34.17	-23.9%	918	0:48	45.90	2.2%	403	0:17	32.08	-28.5%	5000	3:57				
Winter	Chiller Elec. Cost ($)	4.40	1.78	-59.5%			5.67	28.9%			0.68	-84.5%						

Table 2

Comparison of Internal and External Optimization techniques

		NO Opt.	BCPOL				GenOpt			
		Costs	Costs	Savings	# Iter.	Time	Costs	Savings	# Iter.	Time
Heavy-	Total Cost ($)	310.31	249.62	-19.6%			239.66	-22.8%		
Strong-	Cooling Cost ($)	183.64	122.95	-33.0%	1317	2:12	112.99	-38.5%	466	6:25
Summer	Chiller Elec. Cost ($)	130.81	71.73	-45.2%			58.59	-55.2%		
Heavy-	Total Cost ($)	369.38	378.23	2.4%			393.80	6.6%		
Weak-	Cooling Cost ($)	215.48	232.89	8.1%	927	1:50	256.61	19.1%	580	7:36
Summer	Chiller Elec. Cost ($)	152.29	147.22	-3.3%			149.96	-1.5%		
Light-	Total Cost ($)	317.43	268.49	-15.4%			289.77	-8.7%		
Strong-	Cooling Cost ($)	190.75	141.82	-25.7%	1214	1:47	172.10	-9.8%	261	3:38
Summer	Chiller Elec. Cost ($)	136.63	88.00	-35.6%			103.99	-23.9%		
Light-	Total Cost ($)	376.24	364.46	-3.1%			385.06	2.3%		
Weak-	Cooling Cost ($)	222.34	225.94	1.6%	834	1:15	241.77	8.7%	466	6:48
Summer	Chiller Elec. Cost ($)	157.82	143.39	-9.1%			151.28	-4.1%		

Appendix A. Introduction of *EnergyPlus*

Basic Structure of *EnergyPlus*

EnergyPlus is constructed as "loop-in-loop" structure, as illustrated in Figure A-1. When a simulation run is initiated with *EnergyPlus*, the input file is first processed and checked for consistency by calling the subroutine *ProcessInput*. The input values are then stored in global arrays that can be accessed easily by subroutines that are designed to retrieve data from them. Then follows the core line of *EnergyPlus*, i.e., calling subroutine *ManageSimulation* in the *SimulationManager*. Here, an initialization is first carried out and then the "loop-in-loop" simulation is performed.

After the initialization, *EnergyPlus* calculates from outside to inside through an environment loop, a day loop, an hour loop, and a timestep loop. The term "environment" in *EnergyPlus* can be simply viewed as a thermal history concatenated run period. A simulation with one run period (for example, from July 20 to November 20) has one environment. However, a simulation with two run periods (for example, one run period from July 20 to August 30 and another run period from September 1 to December 5), has two environments. By calling subroutine *GetNextEnvironment* in the *WeatherManager* file, the environment loop counts the environments to be simulated and initializes the next available environment to be simulated by the day loop. Each environment contains one or more days. These days are simulated sequentially in the day loop. In each day, 24 hours are simulated from 1 to 24 in the hour loop. In each hour, a timestep loop of up to six timesteps per hour (i.e., 10 minutes) is allowed. The calculation routines of *EnergyPlus* are performed for each timestep. In particular, building zone loads, HVAC system loads, heating/cooling plant loads and eventually energy consumption are calculated for each timestep.

In standard *EnergyPlus*, operating and setpoint schedules are provided as fixed values and are not modified during the simulation. In *EnergyPlus* with an internal optimization module, schedules can be modified by the optimization algorithm. Then, these schedules are fed back to *EnergyPlus* calculation routines to search for the optimal solution.

Building Control Strategies in *EnergyPlus*

EnergyPlus simulates the operation of HVAC systems using a set of schedules. In the input file, week schedules as well as day schedules are defined for the control variables (such as temperature setpoints). For simulations with timesteps less than one hour, hourly values from the day schedules are used to interpolate values for each timestep. At the start of the initial timestep, subroutine *ProcessScheduleInput* is called. This subroutine extracts information about all of the schedules defined in the input file and stores them in a global array, called *Schedule*. Then, subroutine *GetCurrentScheduleValue* is called to determine the schedule values at each timestep. An index for each schedule is defined as the sequence number of that schedule as stored in the *Schedule* array. Knowing the name of the schedule, the schedule index number of a schedule can be retrieved by calling subroutine *GetScheduleIndex*. These schedule-handling subroutines provide a convenient procedure to retrieve schedule values for any timestep by simply knowing the name of the schedule.

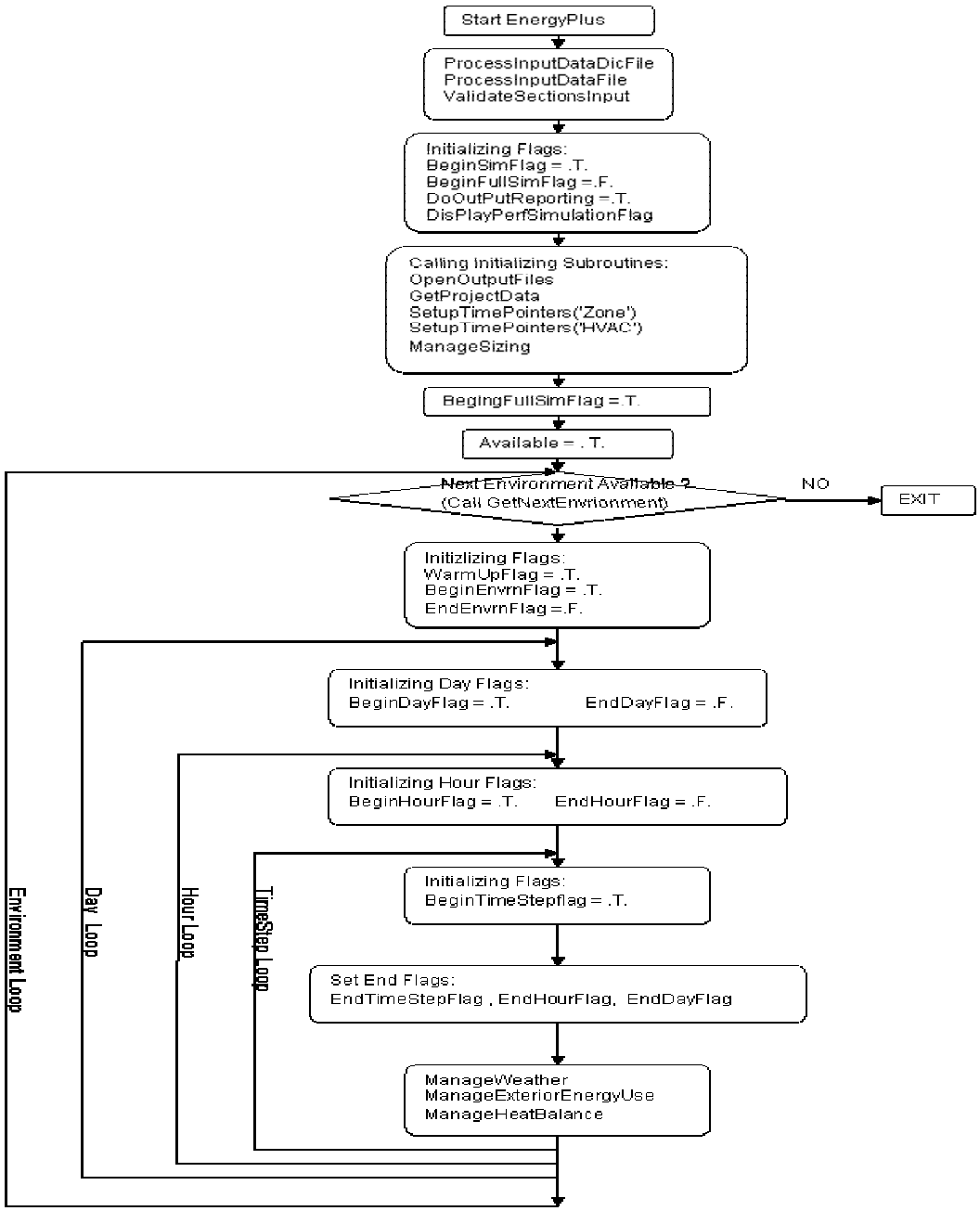

Figure A-1: EnergyPlus v1.01 flowchart

Appendix B. Introduction of Optimization Algorithms

Nelder-Mead Simplex Method

This method, developed by Nelder and Mead (1965), is a multi-dimensional direct search optimization technique that requires only objective functions evaluations without derivatives. This characteristic of the method makes it suitable for optimization for controlling TES systems since it is computationally too expensive to estimate derivatives of the objective function using *EnergyPlus* program. An algorithm using this method is available in the standard texts on numerical recipes. A program based on the Nelder-Mead simplex method, called *BCPOL*, is also provided in the *ISML* library.

Quasi-Newton Method

The *BFGS* method, a popular variant of the Quasi-Newton method, is an optimization method that calculates gradients for the objective function (Broyden, 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970; Rao, 1996). The *BFGS* method can handle optimization problems subject to simple bounds on the variables such as the problem of optimal controls of TES systems. The main disadvantage with the *BFGS* technique is may reach non-optimal solutions due to inaccurate estimates of the objective function gradients. An algorithm using the *BFGS* optimization technique, called *BCONF*, is available in the *ISML* library.

Simulated Annealing

The simulated annealing method has been developed using the analogy of how metals cool and anneal (Belegundu and Chandrupalta, 1999). It can be a useful optimization method when a desired global minimum is among several other local minima. The primary advantage of simulated annealing method is that it can avoid undesirable local minima other techniques tend to get stuck in. It often converges to a solution that is close to the true minimum solution. The main disadvantage of the simulated annealing method is that it is a slow technique and requires substantial computational time.

Population-Based Method

This meta-heuristic method is a type of genetic algorithm, which harnesses the theory of natural selection (i.e., "survival of the fittest"). The algorithm used in the context of this article, called *OptQuest* and developed by Glover and Laguna (2000), utilizes "tabu" search and scatter search to solve non-smooth optimization problems of up to 5,000 variables and 1,000 constraints. Compared to conventional genetic algorithms, *OptQuest* makes greater use of strategic choices and less use of randomization. Where classical optimization methods keep track of a single "best solution" during a search process, *OptQuest* maintains a set of candidate solutions. Any member of the set can lead to a new and better solution, possibly far away from the "best solution" found at a given stage of the search. Because of this feature, *OptQuest* is unlikely to become "trapped" in a local optimal solution.