ELSEVIER

Energy and Buildings 33 (2001) 309–317

ENERGY AND BUILDINGS

www.elsevier.com/locate/enbuild

Efficient solution strategies for building energy system simulation

Edward F. Sowell[a,*], Philip Haves[b]

[a]California State University, Fullerton, CA 92834, USA
[b]Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

Abstract

The efficiencies of methods employed in solution of building simulation models are considered and compared by means of benchmark testing. Direct comparisons between the Simulation Problem Analysis and Research Kernel (SPARK) and the HVACSIM+ programs are presented, as are results for SPARK versus conventional and sparse matrix methods. An indirect comparison between SPARK and the IDA program is carried out by solving one of the benchmark test suite problems using the sparse methods employed in that program. The test suite consisted of two problems chosen to span the range of expected performance advantage. SPARK execution times versus problem size are compared to those obtained with conventional and sparse matrix implementations of these problems. Then, to see if the results of these limiting cases extend to actual problems in building simulation, a detailed control system for a heating, ventilating and air conditioning (HVAC) system is simulated with and without the use of SPARK cut set reduction. Execution times for the reduced and non-reduced SPARK models are compared with those for an HVACSIM+ model of the same system. Results show that the graph-theoretic techniques employed in SPARK offer significant speed advantages over the other methods for significantly reducible problems and that by using sparse methods in combination with graph-theoretic methods even problem portions with little reduction potential can be solved efficiently.

Keywords: Building energy systems; HVACSIM+ models; SPARK models; HVAC simulation; Computational efficiency; Graph theory applications

1. Background

Detailed simulation of building energy systems involves the solution of large sets of non-linear algebraic and differential equations. These equations emerge from component-based simulators such as TRNSYS [1] or HVACSIM+ [2], or equation-based tools such as SPARK [3] or IDA [4]. Since each of these tools employs a different solution strategy, the question arises as to which strategy is most appropriate for the kinds of equations encountered in the building simulation domain.

TRNSYS and HVACSIM+ are both based on subroutines containing algorithmic models of the underlying physics for the represented building system component. TRNSYS, the program with the longest and perhaps most wide spread usage, employs a ''block iterative'' strategy, calling the component subroutines in a sequence largely determined by the order in which they appear in the user's problem definition. Convergence is sought using successive substitution of calculated interface variables into the block inputs on the next iteration. If convergence is indeed obtained, solution is often fast since the number of iteration variables is small and there are no vector-matrix operations. However, the successive substitution method is unreliable in general, so convergence is often slow or not obtained at all. The HVACSIM+ program, which is much like TRNSYS at the problem definition level, assembles a vector of the interface variables throughout the model and employs a Newton-like solution strategy. The advantages sought with this approach are robustness and efficiency, since the information in the Jacobian allows calculation of a better next guess than the previous value alone. Indeed, provided that initial values of the interface variables are within the radius of convergence, the solution is approached quadratically. However, HVACSIM+ often is less efficient than TRNSYS in practice because of the need to calculate the Jacobian and solve the linear equation set that it represents at each iteration. Because no reduction is attempted, the size of this set is the total number of block interface variables, n_i and solving it is $O(n_i^3)$. Consequently, the more rapid and robust convergence can be overwhelmed, resulting in the longer runtimes often experienced relative to an equivalent TRANSYS model.

The IDA and SPARK modeling environments represent a new departure in that they formulate the model and its solution, in terms of equations rather than the algorithmic subroutines employed in TRNSYS and HVACSIM+. One

* Corresponding author.
E-mail address: sowell@fullerton.edu (E.F. Sowell).

0378-7788/01/$ – see front matter
PII: S0378-7788(00)00113-4

Nomenclature

c_i	scalar constant
δ	correction in Newton–Raphson iteration
f	vector of functions being solved in Newton–Raphson iteration
J	Jacobian matrix in Newton–Raphson iteration
LU	lower/upper matrix factorization
n	number of equations and variables
O($f(n)$)	order of notation (the operation in question is bounded from above by $g(n)$, where n is size of data operated on)
PI	proportional–integral control algorithm
$q_{si,j}$	heat source rate per unit surface node in discrete form of Laplace's equation
$T_{i,j}$	temperature of (i, j) node in discrete form of Laplace's equation
U	conductance
$\overline{x}$	solution vector of size n in Newton–Raphson iteration
x_i	scalar variable

advantage of this approach is that the models of individual components are input/output free. That is, the same component model can be used for a variety of different input and output designations. This allows conceptual separation of the *model* from the *problem*, the model is general, and a specific problem is defined only when a specific set of inputs is designated. Although, the two modeling environments are similar in this important respect, the solution methods employed are radically different. In IDA, the equations are formed as residual formulas, e.g. $R = f(x, y, z)$ and R is forced to zero at the solution point. Residual equations comprising math models of individual physical component are grouped into component models, with variables relevant at the system level exposed to the interface. An IDA system model consists of a set of such component models together with set of *coupling equations* that, in effect, equate equivalent interface variables at different component models. The coupling equations are all linear, of the identification form $p_1 - p_2 = 0$ or the conservation form $m_1 + m_2 - m_3 = 0$, but are large in number so that IDA equation sets tend to be quite large and sparse. For example, a simple example used in IDA reports [5] has 26 coupling equations augmenting 12 model equations. An innovative solution strategy employing sparse matrix methods in a Newton-like iterative process is used to solve the resulting large, sparse system. Because the size added by the coupling equation set is an obvious detriment to overall solution efficiency, the solver has a ''compact solution'' option for which the coupling variables and equations are, in some sense, removed. However, the expected theoretical performance improvement is not realized in the implementation, as it appears to in fact *decrease* solution speed [5]. Nonetheless, there is some anecdotal data (from informal discussions with users) to suggest that IDA may be somewhat faster than HVACSIM+ on some problems. Unfortunately, no benchmark testing results have been reported for IDA, so the actual performance remains uncertain.

Like IDA, SPARK [3] is equation-based. However, SPARK relies upon the mathematical graph for model representation and solution rather than the matrix. To support the graph, rather than expressing equations as residuals, they are expressed in the form $x = f(y, z)$, where the functions are symbolic inverses of the user-supplied model equations. This allows graph algorithms to be used to determine a sequence of function evaluations that leads to the solution. This alone is an advantage, since it eliminates the need for coupling equations entirely. Further, it allows the problem to be decomposed into separately solvable (i.e. strongly connected) components. Within each strong component, if no direct sequence is possible, as evidenced by a cyclic problem graph, a small ''cut set'' is determined so as to minimize the number of variables involved in the subsequent Newton-like iteration. As a result of these reductions, the size of the Jacobian matrix, and hence, the linear set that must be solved at each Newton iteration, is reduced, often significantly. Consequently, as will be shown in this paper, solution speed is greatly reduced.

While ideas from graph theory have been used in connection with equation system solving before [6–12], SPARK applies graph methods directly to the non-linear equations. The graph, rather than the matrix, is the primary data structure for storing the problem structure and data and as already noted, graph algorithms are employed to determine a solution sequence that operates directly on the non-linear equations. Another distinctive attribute of the SPARK approach is that the model equations are stored individually, rather than packaged into modules and are treated as equations rather than as formulae with assignment (algorithms). Symbolic methods are employed to find explicit inverses of the equations, when possible, to ensure computational efficiency. In these ways SPARK is unique. However, increasingly, simulation software is employing some of the ideas embodied in SPARK. For example, Klein, in collaboration with F. Alvarado, produced the Engineering Equation Solver [13], which employs decomposition using sparse matrix methods. This is conceptually the same as the strong component decomposition done in SPARK. However, reduction within blocks is not done in this software. In addition, TRNSYS has recently been modified to allow ''reverse solving'' [1]. This is a move toward input/output free (non-algorithmic) modeling, another tenet of SPARK. Also in the building context, Tang has applied graph-theoretic methods to improve matrix-based solution schemes [14,15].

Although, the SPARK methodology is well established, there has been relatively little systematic comparison of solution speed between SPARK and alternative methods available for solving large sets of equations, such as arise in building simulation. In order to begin to fill this gap, a simple benchmarking experiment was designed. Two problems sets were defined: (a) a replicated set of four

non-linear equations, and (b) the Laplacian equation, i.e. heat conduction, in a two dimensional grid of various sizes. These two problems, while somewhat removed from mainstream building system simulation, were selected to represent the endpoints in the degree to which problems are suited to the methods used in SPARK. To complement findings from these simple problems, the study was extended to include actual building HVAC systems models of considerable complexity. The system selected is one previously studied by Haves [16] using a different building simulation tool, thus, providing opportunities for direct comparison. This work was reported at the Building Simulation '99 Conference [17] in Kyoto.

Discussion at the Kyoto conference raised the question of how effective the SPARK graph-theoretic methods are when compared to state-of-the-art sparse matrix solution methods applied directly to the unreduced problem, as is done in IDA. Specifically, are the techniques routinely applied in sparse matrix packages fully equivalent to SPARK methods, thereby, making it unnecessary to carryout graph-theoretic reduction directly on the non-linear problem? To address this question, one of the problems reported in the Kyoto paper was solved using the sparse matrix package used in IDA, SuperLU [18]. Because SuperLU appears to be one of the most advanced sparse matrix packages, it would seem that if the answer to the question is positive, then it should solve faster than the SPARK implementation. This is shown not to be the case. That is, the new results confirm that the SPARK methods, at least for problems with significant reduction potential, are significantly faster than sparse methods alone. In addition to these new results, more discussion is provided on the comparison to HVACSIM+. Otherwise, the results here are the same as in the Kyoto paper and are presented here for the convenience of the reader.

2. Non-linear equation example

The first benchmark problem derives from a problem in the SPARK User's Manual consisting of four highly non-linear equations:

$$x_1 + x_3 + x_2^2 + \sqrt{x_2} = c_1, \qquad x_2 = x_1\,\mathrm{e}^{x_1},$$
$$x_1x_4 + x_3x_4 + x_4^3 = c_2, \qquad x_4 = x_3\,\mathrm{e}^{-x_3} \tag{1}$$

SPARK finds a solution to these equations by the calculation sequence:

$$x_3 = 0.1, \qquad x_4 = x_3\,\mathrm{e}^{-x_3}, \qquad x_1 = \frac{c_2 - x_3x_4 - x_4^3}{x_4},$$
$$x_2 = x_1\,\mathrm{e}^{x_1}, \qquad x_3 = c_2 - x_1 - x_2^2 - \sqrt{x_2}, \quad \text{iterate on } x_3 \tag{2}$$

Using the default SPARK solution process, Newton–Raphson iteration is performed until the difference between two successive values of x_3 is less than a specified tolerance. Thus, it is seen that reduction of 4:1 is achieved relative to conventional practice of iteration on all unknown variables. With $c_1 = 3000$ and $c_2 = 1$, the solution found is $x_4 = 0.288576$, $x_1 = 2.9273$, $x_2 = 54.6738$ and $x_3 = 0.454716$.

For the numerical experiments, this set of equations was implemented as a SPARK macro class called *example* which was then instantiated $n/4$ times to get a problem of size n. Obviously, every instance of *example* is in fact a separately solvable problem. SPARK is able to discern this structural regularity and partition the problem graph into $n/4$ strongly connected components, each with a cut set of size one. Consequently, during the numeric phase of the solution, $n/4$ single-variable iterative solutions are carried out. Most conventional, general solvers would instead solve a single equation set of size n by iteration on all n variables. If Newton–Raphson iteration were used, a linear set with an n by n coefficient matrix would need to be solved at each iteration.

For comparison purposes, this equation set was also solved with three other methods. First, a hand-crafted Newton–Raphson non-linear solver (*nlsolve*) was written specifically for this equation set, with the problem size as an input parameter. In this solver, the four Eq. (1) were coded in a single function that was called as needed for calculation of the residual functions and the Jacobian. The matrix functions from SPARK were used to calculate the Jacobian numerically and solve the linear set for new estimates of the iteration variables. Second, a sparse Newton–Raphson solver (*spnlsolv*) was written using the sparse LU solve function from the Meschach sparse matrix package [19]. The interface, function evaluation, Newton–Raphson loop and output were basically the same as for *nlsolve*, with the only difference being the use of sparse storage date types and sparse matrix solver functions from Meschach. The same solution tolerance (1×10^{-6}) was used in both cases.

Comparative run-times with a 333 MHz AMD K-6/2 processor are shown in Fig. 1. As would be expected, the experimental results show $O(n^3)$ performance for the full matrix solution. The solver based on the Meschach sparse matrix functions shows much better performance, approximately $O(n^2)$. Also as expected, SPARK is much better than the sparse implementation, showing about $O(n)$.

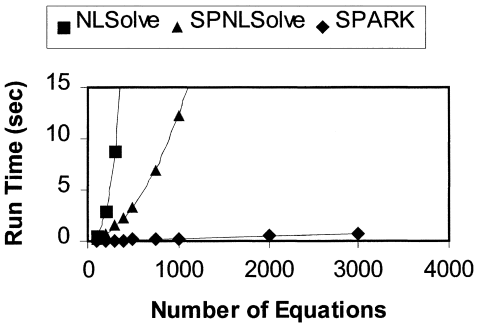

Fig. 1. Solution times for non-linear benchmark, SPARK vs. Meschach.

In order to confirm that these dramatic solution speed improvements are not attributable to the particular sparse package chosen, this problem was also coded for solution with the SuperLU sparse package [18]. This freely available software is the result of US Government sponsored research at the University of California, Berkeley, and is currently supported by the National Energy Research Scientific Computing Center (NERSC) at the Lawrence Berkeley National Laboratory. If not the most advanced software in this category, it appears at least to be the most current. For these reasons, and also because it supports parallel computation models, it was chosen by the IDA development team for use in their program.

The results of this substudy are presented in Fig. 2. For comparison purposes, the SPARK and Meschach results are replotted in the figure. Thus, the uppermost curve in the figure is for the Meschach solution, while the lowest one is for SPARK. The middle curve is for the SuperLU solution. The upshot is that, although, SuperLU is somewhat faster than Meschach (15 versus 48 s for $n = 2000$), SPARK is much faster. More importantly, it is clear that SuperLU, like Meschach, is performing at $O(n^2)$ as compared to $O(n)$ for SPARK.

In order to explore the full potential of the SuperLU package, several solution options were tried. In particular, each of the three column ordering options was tried and found to have no noticeable effect on run-time. The option to reuse the factoring information from the initial iteration rather than factoring from scratch at each Newton iteration was also tried, again without significant effect. Finally, to see if initial equation ordering had an impact, the driver program was modified to randomize this ordering in constructing the Jacobian. Again, there was no significant effect. To demonstrate these findings, the results for the natural ordering and without refactorization and random ordering of the equations is overlaid with the case showing the greatest departure. The effects are so small as to be unnoticeable in the plot. The explanation for these findings is presented below (see Section 5).

3. Laplace's equation example

The second benchmark problem, purposely chosen to be not well suited to the SPARK methodology, is Laplace's equation in two dimensions. This equation models many physical phenomena, including heat transfer in a thin, square plate with uniformly distributed heat source and uniform boundary temperature. The problem is discretized by dividing the square into a uniform grid of specified size. Each cell in the grid is represented by a nodal temperature $T_{i,j}$ and is governed by a heat balance equation:

$$q_{si,j} = (T_{i,j} - T_{i,j+1}) + (T_{i,j} - T_{i-1,j}) + (T_{i,j} - T_{i,j-1}) + (T_{i,j} - T_{i+1,j}) \quad (3)$$

where $q_{si,j}$ is the heat source rate per unit surface area. As can be seen, the internodal conductance is assumed to be 1.0.

This problem is coded for sparse solution in the Meschach tutorial [19]. However, for this study, that implementation was modified to employ sparse LU factorization, since the use of Cholesky factorization and sparse conjugate gradient iteration in the original code applies only to symmetric positive definite matrices, a condition satisfied by the Laplacian but not often found in general simulation problems.

For comparison, a program was written to generate SPARK problem and input files for the same equation system. The grid size was varied between 3 and 45, yielding equation set sizes between 9 and 2025. Both SPARK and the Meschach-based solver were compiled with the same compiler and optimization options.

In the initial SPARK implementation, each grid node was represented with a SPARK macro object called *node* constructed with atomic *conductor* and *sum* objects from the

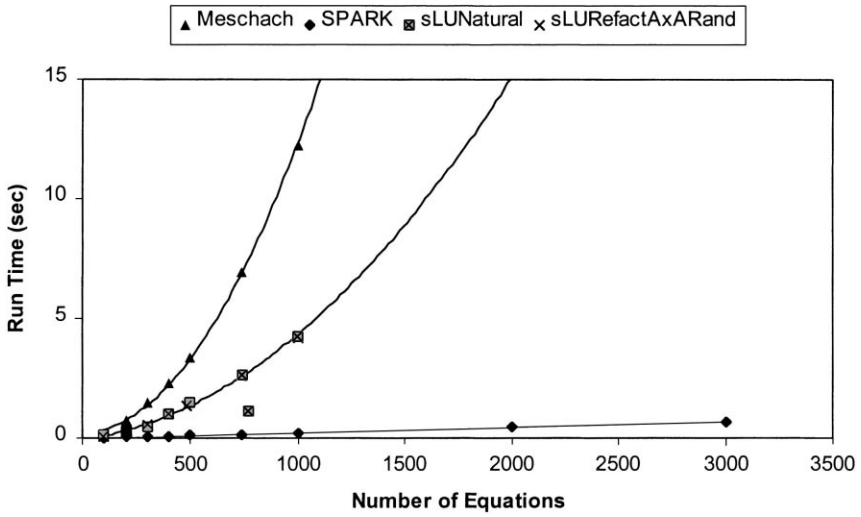

Fig. 2. Solution times for non-linear benchmark, SPARK vs. SuperLU.

SPARK HVAC class library [20]. With this implementation and a grid size of 19 × 19, the SPARK solution time was about 60 times that of the Meschach solver. While a weak SPARK showing was anticipated for this problem, this huge difference was a surprise, calling for further investigation.

The first reason for the long SPARK run-times was found to be representation of the node as a macro object. This resulted in seven distinct equations for each node, four of the form $q = u(T_2 - T_1)$ and three of the form $a = b + c$, giving 2530 equations for the grid size of 19 × 19. Although, the SPARK graph-theoretic algorithms were able to find a cut set of 342, a reduction of 86%, the Meschach implementation was hand-crafted so that there were only 361 equations in the set to be solved. Moreover, the Meschach implementation assumed an internodal conductance of unity (Eq. (3)), so no multiplications were needed. Therefore, even after graph-theoretic reduction brought the Jacobian size down to approximately the size seen by Meschach, the SPARK model required many more arithmetic operations in evaluation of each equation. In short, the numerical problems seen by the two solvers were not the same, even though, they both represented the same physical problem.

To try to get a more meaningful comparison, both models were changed in several ways. First, the SPARK implementation was revised to more closely approximate the problem as seen by Meschach. A specialized SPARK atomic object class was written to represent the node as a single heat balance equation with an assumed unit conductance, as in the Meschach implementation. With this revision there were only 361 objects in the SPARK model for the 19 × 19 grid, and the SPARK solution times improved considerably.

Then, to see to what extent the presumably more efficient data handling methods in Meschach contribute to its speed advantage, the SPARK solver was modified to optionally use either sparse or non-sparse vector-matrix data structures and functions from Meschach when updating the solution vector. These changes both produced substantial speed-up, with the sparse handling option performing essentially as well as Meschach (see Section 5).

Another difference observed between the two approaches was that because SPARK is a general non-linear solver, it employs Newton–Raphson iteration, requiring numerical calculation of the Jacobian matrix at each iteration. In contrast, the hand-crafted Meschach model is aware of the problem linearity and constant coefficients and consequently sets up the conduction matrix only once, directly from the given coefficients. Since, this study was concerned principally with solving methods for non-linear equation systems, it was of interest to see how much of the run-time difference was due to extra work in SPARK associated with non-linear solving. However, rather than changing SPARK, a second Meschach-based model was developed in which the system of equations was set up for solution as if they were non-linear. That is, a Jacobian was formed numerically, as in SPARK, and Newton–Raphson iteration was performed to obtain a solution. A Meschach sparse solver and supporting vector-matrix routines were used to calculate the solution vector for each iteration. Note that, while this approach has the advantages of Meschach's efficient data handling and sparse matrix operations, it does not share SPARK's ability to reduce the Jacobian size.

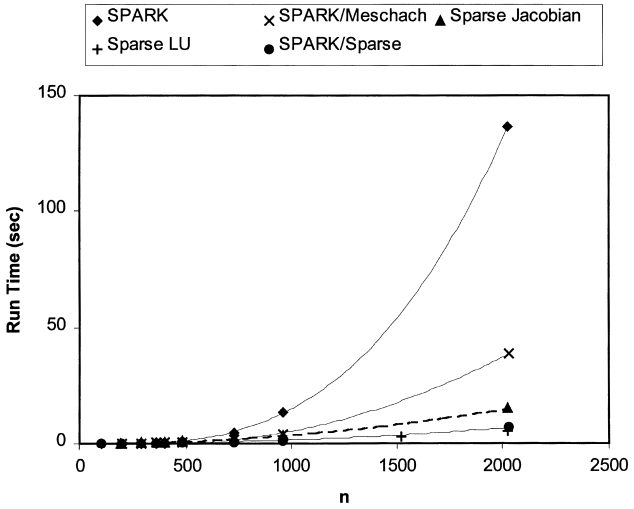

Fig. 3. Solution times for the Laplace's equation example.

The performance of the various solution methods is summarized in Fig. 3. The three solid curves show SPARK solution times versus total number of equations. The uppermost curve is for solution with the current, standard SPARK. The next lower curve was generated using the modified version of SPARK with the Meschach non-sparse handling of the Jacobian as mentioned above, while the lowest curve results from use of the sparse option. In all three cases, the graph-theoretic matching and cutting were coerced by selecting input options so as to get the theoretical minimum cut set size while preserving diagonal dominance of the reduced Jacobian. This is an important qualification and is discussed further below.

The dash-line curve in Fig. 3 is for solution using the Meschach-based Newton–Raphson solver described previously. Performance is seen to be significantly better than the standard SPARK, and somewhat better than the modified SPARK using non-sparse methods. However, it is not as good as SPARK using sparse Jacobian handling.

The final results in the figure are for the Meschach tutorial program using sparse LU decomposition. These results overlay almost exactly those for the modified SPARK using sparse Jacobian handling, so a separate trend line is not plotted. However, this agreement is coincidental. Apparently, the reduced Jacobian size in SPARK offers a speed advantage that overcomes SPARK overhead costs such as function calls and numerical Jacobian evaluation, which are not done in Meschach.[1]

[1] In the current implementation, SPARK makes a call to a C++ function for every equation evaluation.

4. HVAC benchmarks

Going beyond simple benchmark examples, the numerical methods used in SPARK were also evaluated by modeling an airflow system employing discrete-time controllers. The example used was a typical HVAC airflow network and its associated control loops, a problem involving significant computational burden [16].

A number of steady state component models were implemented as SPARK objects, including variable speed centrifugal fans, flow diverters, flow mixers and control dampers. In modeling air flow, a square law dependence of total pressure drop on flow rate was used above a critical flow rate and a linear dependence was used below the critical flow rate to avoid known computational problems with air at low flow rates. Dynamic models included flow sensors, pressure sensors, rate limits, discrete-time proportional-plus-integral (PI) controllers and fan control strategies based on PI control.

Fig. 4 shows the system that was simulated. The positions of the mixing box dampers determine the proportions of outside and recirculated air that are filtered and cooled before being supplied to the six zones of the building. The positions of the terminal box dampers determine the air flow rates to the corresponding zones. The speed of the supply fan is determined by a PI controller that regulates the static pressure of the air in the supply duct. The speed of the return fan is determined by a PI controller that regulates the difference between the supply airflow rate and the return air flow rate. For the purposes of the benchmark tests, the various damper openings were treated as boundary conditions. In the airflow network used to model the duct system there were 28 flow rate variables and 30 pressure variables, three of which were boundary variables.

In order to assess the benefits of using SPARK methods, a base case and two reference cases were constructed. The base case was modeled with SPARK in the normal manner, allowing the graph-theoretic techniques to perform reduction of the problem graph. The two reference cases were:

1. The system modeled using the HVACSIM+ program [2], as in the previous work [16].
2. The system modeled using SPARK, but inhibiting the normal problem reduction techniques.

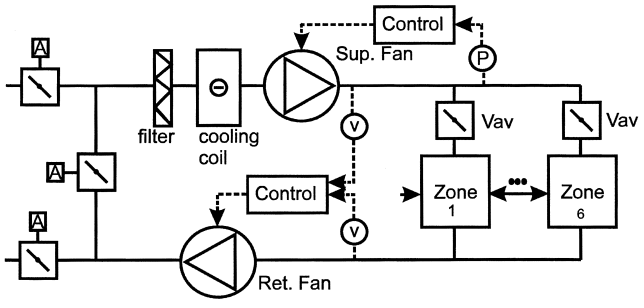

Fig. 4. HVAC system.

Table 1
Comparison for HVACSIM+ and SPARK

Control loops	Time (s)		Iteration variables	
	HVACSIM+	SPARK	HVACSIM+	SPARK
Intact	1135	48.8	62	15
Broken	785	52.7	55	15

The use of the two reference cases enables the benefits of the graph-theoretic techniques to be separated from the effects of program architecture. For all three cases, the simulation problem was a series of set-point changes for each controller followed by a disturbance caused by progressive closing of the VAV terminal boxes.

In addition to these comparisons directed at assessment of the importance of reduction, a side study was performed to determine whether ''breaking'' of control loops offers computational advantage. The interest in this derives both from the needs of proper models of discrete time sample-and-hold controllers, and from the introduction of artificial delays as a computational device to speed solution.

Comparisons between HVACSIM+ and SPARK are shown in Table 1. In the first comparison, 'control loops intact', the flow network equations and the controller equations are solved simultaneously. The main result is that SPARK is 15–20 times faster than HVACSIM+. The obvious reason for the speedup is that SPARK achieves a 4:1 reduction in the number of variables in the iteration vector.

In the second comparison, control loops broken, the set of simultaneous equations representing the airflow network and those representing the control system are solved sequentially. This corresponds to breaking the algebraic loops, such as by introduction of a sample-and-hold in the controller or an artificial delay. Whereas, a significant benefit was gained from breaking the control loops when using HVACSIM+, there was no such benefit when using SPARK. The reason for this, as discussed in another paper [21], is that SPARK finds fan discharge pressures of the supply and return fans to be good choices for break variables, so the computation loops in question are broken regardless.

In order to determine how much of the SPARK advantage can be attributed to the problem reduction, these techniques were disabled, producing the results shown in Table 2 for the intact loops case. These results show that the effect of the problem reduction techniques in SPARK is to speed the benchmark problem up by a factor of 13. This is

Table 2
Effect of SPARK reduction

	HVACSIM+	SPARK unreduced	SPARK reduced
No. of equations	62	62	15
Execution time (s)	1135	637	48.8

approximately what would be expected from the reduction in the number of equations.

To investigate further the question of whether the reduction of ~4:1 in the number of problem variables in the iteration vector observed in this example is likely to be achieved in other HVAC simulation problems, a number of series/parallel network configurations representative of HVAC airflow networks were subjected to the SPARK problem reduction techniques. Twenty configurations were studied, with between 2 and 24 flow elements and up to eight parallel paths. In 15 cases, the size of the cut set, and hence the number of iteration variables, was equal to the number of parallel flow paths. In four cases, the size of the cut set exceeded by one the number of parallel flow paths, and in the remaining case it exceeded it by two. The ratio of the numbers of equations in the unreduced and reduced problems ranged from 3.0:1 to 5.8:1, with a mean value of 4.4:1, a similar value to that found in the example case described above.

The question of whether control loops typically add to the number of equations in the reduced problem was addressed by considering a further six configurations that each included one control loop. In five out of the six cases, adding the control loop did not increase the size of the cut set in the reduced problem, and in one case it increased the size of the cut set by one. The interpretation of these results is discussed in [21].

5. Discussion

The above results confirm that the SPARK methodology offers significant reduction in solution times relative to both conventional and sparse matrix methods in the solution of certain kinds of non-linear equation systems. This is borne out most dramatically by the contrived non-linear benchmark problem, but is also quite clear from the HVAC control application. However, in the case of the example involving Laplace's equation, we observe that without some user intervention, SPARK has difficulty competing with sparse methods. Understanding why this occurs is important in order to guide improvement of the SPARK methods and to delineate properly the class of problems amenable to SPARK methods.

To understand the observed differences in run-times, it is important to note that at the heart of the Newton–Raphson non-linear solution process is the solution of a linear problem. That is, during each iteration the solution vector must be updated by solution of the equations:

$$J\delta = f(\bar{x}^k), \qquad \bar{x}^{k+1} = \bar{x}^k + \delta \tag{4}$$

where $\bar{x}$ is the solution vector of size n, f the vector of functions being solved, δ the correction vector and J is the Jacobian. Now, since J is $n \times n$, calculation of its elements is $O(n^2)$, whether done numerically by finite difference (the usual case), or from derivative formulae. Moreover, solution of linear systems is in general an $O(n^3)$ process. Since, evaluation of the functions f is only of $O(n)$, evaluation of the Jacobian and solving the linear set are the overriding factors in determining run-time. Consequently, anything that can be done to reduce the size of the Jacobian has a powerful effect, especially for large problem size.

SPARK gains its advantage over conventional methods by reducing the Jacobian size. It does this in two, separate ways: *decomposition* and *cut set reduction*. Decomposition is possible when the equation set is, in reality, a sequence of separately solvable problems. SPARK is able to detect this property automatically and carry out the decomposition without intervention. For example, the non-linear benchmark problem with 100 equations and variables is decomposed into 25 subproblems (or in graph terms, strongly connected components) each of size four. This alone would reduce the run-time from $O(100^3)$ to $25 \times O(4^3)$, i.e. a factor of 625. Cut set reduction refers to reducing the sizes of the Jacobians of the subproblems. This is done by an algorithm that finds a small set of nodes in the problem graph that breaks all cycles, called a cut set. The cut set variables then form the iteration vector for the Newton–Raphson process. Again, looking at the non-linear benchmark problem, a cut set of size one was discovered in each component. Thus, the 25 Jacobians are all 1×1, so the overall theoretical run-time reduction is by a factor of 40,000. This efficiency gain is only partially realized due to the overhead associated with the SPARK implementation, but this analysis clearly explains the observed excellent performance for this example.

A similar analysis shows why SPARK has difficulties with the Laplacian example. In this case, the problem graph (Fig. 5) is more complex, with each node biconnected to four neighbors. One consequence of this high degree of interconnectedness is that the problem does not decompose, so that it has to be solved as a single strongly connected component. Another is that a small cut set is hard to find. The normal SPARK cut set algorithm works on the principle of contraction, in which nodes with single incoming or outgoing edges are bypassed and removed, thereby, producing progressively simpler graphs from which the cut set can be deduced. However, there are no such nodes in this graph, so the algorithm must revert to arbitrary removal of nodes into the cut set [22]. In many problems, arbitrary removal results in further opportunities for contraction. Such is not the case

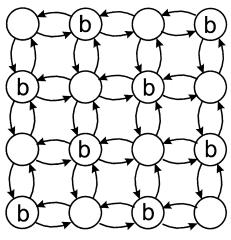

Fig. 5. Graph for Laplace's equation example.

here, so the algorithm continues to do arbitrary removal, arriving at a relatively large cut set. For example, in the 45×45 grid case (2025 nodes) the discovered cut set is 1894. This is a reduction in Jacobian size of only 5%, hardly enough to overcome overhead costs. Indeed, with this cut set the SPARK run-time was nearly seven times that shown in Fig. 3. However, it is not difficult to see that a much smaller cut set is possible for the Laplace's equation example. Suppose that for *odd* rows in the grid, we mark break (b) on every *even* column node, and apply the reverse policy in *even* rows. This creates a checkerboard pattern on the grid in which every marked node is surrounded by unmarked ones, as shown in Fig. 5. Clearly, the marked nodes form a cut set, since every unmarked node can be calculated given temperature values at the marked ones. This policy can be implemented in a SPARK model using the *break_level* keyword, coercing the algorithm to choose the wanted breaks. When this is done, the cut set size is $n/2$, producing results shown in Fig. 3. In a future version of SPARK, it may be possible to improve the matching and cutting algorithms to detect regularities in the problem graph so as to automatically arrive at smaller cut sets in problems of this nature.

While SPARK seeks solution efficiency through graph-theoretic reduction, sparse solvers seek it by taking advantage of sparsity in the Jacobian. The first goal in a sparse implementation is to reduce memory usage by storing only non-zero elements in matrices. Secondly, special functions are used to carry out operations such as vector-matrix multiplications with operations performed only on non-zero elements. The Meschach package is very effective in this regard, as evidenced by its performance on the Laplacian problem here. Indeed, the solutions times, shown in Fig. 3, are not only (slightly) smaller than the best SPARK performance, but also are of $O(n^2)$. The reason is that, regardless of the size of the matrix, there are only five non-zero entries in each row, and consequently only five multiplications and four additions in the evaluation of each row-vector product. That is, the per-row operations are constant rather than $O(n)$.

It is sometimes claimed that sparse matrix methods routinely do the equivalent of SPARK's graph-theoretic reduction. The results presented here show clearly that this claim is not true. SuperLU, arguably a state-of-the-art sparse package, can do no more than discover and take partial advantage of the natural block diagonal structure of the non-linear benchmark problem studied here. This is because the most commonly used row–column permutation strategies (including all those available in SuperLU) are aimed at reducing fill during LU decomposition rather than any equivalent of the reduction available by graph-theoretic operations directly on the non-linear problem. Nonetheless, we are well aware that more advanced sparse implementations go beyond memory saving and efficient vector-matrix operations. For example, there are algorithms that, if possible, reorganize the matrix into block-diagonal form, allowing a partitioned solution that is entirely equivalent to the strong component decomposition done in SPARK [7,13]. However, neither the Meschach nor the SuperLU package currently has this feature, as evidenced by their rather poor performance in our non-linear benchmark example. Therefore, to be competitive with graph-theoretic methods, non-linear solvers using sparse matrix packages must go somewhat beyond merely calling the linear solvers built into the sparse packages. While we cannot assert that sparse matrix-based solvers could not be adapted to incorporate these ideas, we do claim that they are not routinely applied, if they are indeed applied at all, in software currently in use in the building simulation domain.

In spite of these arguments, we are not prepared to entirely reject sparse methods. Indeed, an important outcome of this study is the importance of employing sparse methods within SPARK. This is because in problems such as the Laplace's equation example, the Jacobian can still be quite sparse, even after reduction. In the 45×45 grid, only 1% of the 1012×1012 Jacobian cells is non-zero. This explains the dramatic reduction in SPARK run-time in Fig. 3 for the sparse Jacobian modification. Work now underway will provide a sparse solution option in SPARK. This will be selectable on a component by component basis.

The HVAC simulation benchmarks also provide insights into the effectiveness of SPARK solution methodology. From Table 1, we see that SPARK has a clear advantage over HVACSIM+ in simulation of detailed control models. Table 2 also shows that a good deal of the advantage remains even if reduction is not done, raising questions about what other factors are at play. We are unable to fully answer this question, but some contributing factors might be heavier reliance on preprocessing of the problem in SPARK. That is, the graph-theoretic analysis is carried out in a separate set up program, which then generates a C++ file for compilation. The output of the set up program is an efficient representation of the problem, with the computation sequence effectively built into the data structures. This saves time that a program like HVACSIM+ has to spend moving data from place to place and doing run-time branching checks and control transfers. In a large problem (thousands of equations), there can be a significant computational effort involved in the preprocessing step. For short simulation runs, such as the benchmarks reported herein, the time involved might be comparable with or longer than that required to run the problem. However, the SPARK approach has clear advantages for longer or repeated runs.

6. Conclusions

The principle conclusion that can be drawn from this work is that SPARK outperforms conventional and sparse matrix methods for solution of problems that can be decomposed and/or reduced with graph-theoretical techniques. Roughly speaking, execution time savings will be $O(mr^3)$ where r is the ratio of the largest cut set size to the number of equations in the problem, and m is the number of strongly connected

components into which the problem partitions. Typical HVAC air flow systems simulation models, including associated controls, are among the problems that benefit from the SPARK solution methodology. The reduction techniques produced close to the maximum reduction in the benchmark HVAC problem, and there are indications that similar reductions can be expected in the broad class of problems involving flow networks and their associated control systems. Reductions in execution time of more than an order of magnitude can be expected relative to full-matrix solvers, such as HVACSIM+. While direct benchmarks were not carried out for IDA, our indirect tests suggest that the sparse methods employed in that program will not be comparable to SPARK for problems in this class. On the other hand, problems characterized by a high degree of interconnectivity, such as energy, mass or momentum transport in homogenous media, allow limited reduction and, therefore, are not prima fascia candidates for SPARK solution methods. However, by proper coercion of matching and cut set selection, significant execution time reduction can still be achieved. Finally, since the reduced Jacobian in homogeneous transport problems is still very sparse, conventional sparse matrix methods can be beneficially applied after SPARK reduction. When this is done, SPARK can be competitive with sparse solvers for homogeneous transport problems, and probably superior for system simulations in which reducible and homogeneous transport components must both be solved.

Acknowledgements

Portions of this work were sponsored by the Japan Ministry of Education, the United Kingdom Royal Academy of Engineering Foresight Award Scheme and the Office of Building Technologies, Building Systems Division, US Department of Energy.

References

[1] A. Fiscal, et al., Developments to the TRNSYS simulation program, Journal of Solar Energy Engineering, 123 (1995) (5).
[2] C. Park, D.R. Clark, G.E. Kelly, An overview of HVACSIM+, a dynamic building/HVAC control systems simulation program, in: Proceedings of the First Building Energy Simulation Conference, International Building Performance Simulation Association, Seattle, WA, 3–6 December 1985.
[3] W.F. Buhl, et al., Recent improvements in SPARK: strong component decomposition, multivalued objects, and graphical interface, in: Building Simulation '93, International Building Performance Simulation Association, Adelaide, 1993.
[4] P. Sahlin, A. Bring, IDA solver — a tool for building and energy system simulation, in: Proceedings of the Building Simulation '91, International Building Performance Simulation Association, Nice, France, 1991.
[5] L. Eriksson, G. Soderlind, A. Bring, Numerical methods for the simulation of modular dynamical systems, Royal Institute of Technology (KTH), Stockholm, 1992.
[6] D.W. Edwards, Robust decomposition techniques for process design and optimization, in: Chemical Engineering, University of London, London, 1982, p. 243.
[7] R.E. Tarjan, Depth first search and linear graph algorithms, SIAM Journal of Computing 1 (1972) 146–160.
[8] D.V. Steward, On an approach to techniques for the analysis of the structure of large systems of equations, Society of Industrial and Applied Mathematics 4 (4) (1962) 321–342.
[9] S. Parter, The use of linear graphs in Gauss elimination, Society of Industrial and Applied Mathematics 3 (2) (1961) 119–130.
[10] C.L. Coates, Flow-graph solutions of linear algebraic equations, IRE Transactions on Circuit Theory 6 (1959) 170–187.
[11] F. Harary, A graph-theoretic method for the complete reduction of a matrix with a view toward finding its eigenvalues, Journal of Mathamatics and Physics 38 (1959) 104–111.
[12] F. Harary, The determinant of the adjacency matrix of a graph, Society of Industrial and Applied Mathematics 4 (3) (1962) 202–210.
[13] S. Klein, Engineering equation solver (EES), 1991, F-Chart Software, Madison.
[14] D. Tang, The generalised system solution classes in the EKS environment, in: Proceedings of the Building Simulation '91, International Building Performance Simulation Association, Nice, France, 1991.
[15] D. Tang, J.A. Clarke, Application of the object oriented programming paradigm to building plant system modelling, in: Proceedings of the Building Simulation '93, International Building Performance Simulation Association, Adelaide, 1993.
[16] P. Haves, L.K. Norford, M. DeSimone, A standard simulation testbed for evaluation of control algorithms & strategies, in: Transactions of the American Society of Heating, Refrigerating and Air-conditioning Engineers, 104 (1998) 1.
[17] E.F. Sowell, P. Haves, Numerical performance of the SPARK graph-theoretic simulation program, in: Proceedings of the Building Simulation '99, International Building Performance Simulation Association, Kyoto, 1999.
[18] J.W. Demmel, J.R. Gilbert, X.S. Li, SuperLU User's Guide, 1999, University of California, Department of Computer Science, Berkeley, CA.
[19] D.E. Stewart, Z. Leyk, Meschach: matrix computation in C, in: The Centre for Mathematics and its Applications, The Australian National University, 1994.
[20] E.F. Sowell, M.A. Moshier, HVAC component model libraries for equation-based solvers, in: Proceedings of the Building Simulation '95, International Building Performance Simulation Association, Madison, WI, 1995.
[21] P. Haves, E.F. Sowell, The application of problem reduction techniques based on graph theory to the simulation of non-linear continuous systems, in: Proceedings of the EuroSim, Society For Computer Simulation, Birmingham, England, 1998.
[22] H. Levy, D.W. Low, Contraction algorithm for finding small cycle cut sets, Journal of Algorithms 9 (1988) 470–493.