

# A COMBINATORIAL SCHEME FOR DEVELOPING EFFICIENT COMPOSITE SOLVERS <sup>\*</sup>

Sanjukta Bhowmick, Padma Raghavan, and Keita Teranishi

Department of Computer Science and Engineering  
The Pennsylvania State University  
220 Pond Lab, University Park, PA 16802-6106  
{bhowmick,raghavan,teranish}@cse.psu.edu

**Abstract.** Many fundamental problems in scientific computing have more than one solution method. It is not uncommon for alternative solution methods to represent different tradeoffs between solution cost and reliability. Furthermore, the performance of a solution method often depends on the numerical properties of the problem instance and thus can vary dramatically across application domains. In such situations, it is natural to consider the construction of a multi-method composite solver to potentially improve both the average performance and reliability. In this paper, we provide a combinatorial framework for developing such composite solvers. We provide analytical results for obtaining an optimal composite from a set of methods with normalized measures of performance and reliability. Our empirical results demonstrate the effectiveness of such optimal composites for solving large, sparse linear systems of equations.

## 1 Introduction

It is not uncommon for fundamental problems in scientific computing to have several competing solution methods. Consider linear system solution and eigenvalue computations for sparse matrices. In both cases several algorithms are available and the performance of a specific algorithm often depends on the numerical properties of the problem instance. The choice of a particular algorithm could depend on two factors: (i) the cost of the algorithm and, (ii) the probability that it computes a solution without failure. Thus, we can view each algorithm as reflecting a certain tradeoff between a suitable metric of cost (or performance) and reliability. It is often neither possible nor practical to predict a priori which algorithm will perform best for a given suite of problems. Furthermore, each algorithm may fail on some problems. Consequently it is natural to ask the following question: Is it possible to develop a robust and efficient composite of

---

<sup>\*</sup> This work has been funded in part by the National Science Foundation through grants NSF CCR-981334, NSF ACI-0196125, and NSF ACI-0102537.

multiple algorithms? We attempt to formalize and answer this question in this paper.

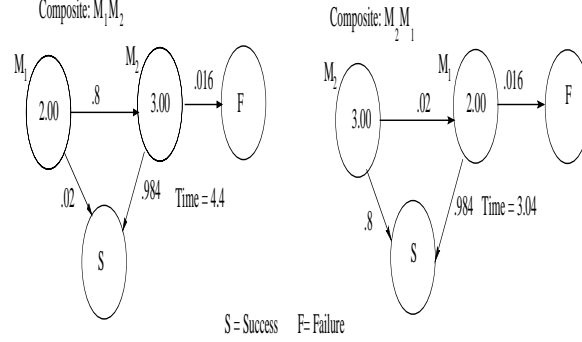
An illustrative example is the problem of solving sparse linear systems. We have a variety of algorithms for this problem, encompassing both direct and iterative methods [3]. Direct methods are highly reliable, but the memory required grows as a nonlinear function of the matrix size. Iterative methods do not require any additional memory but they are not robust; convergence can be slow or fail altogether. Convergence can be accelerated with preconditioning, but that leads to a larger set of preconditioning methods in addition to the basic iterative algorithms. In such cases there is often no single algorithm that is consistently superior even for linear systems from a specific application domain. This situation leads us to believe that rather than relying on a single algorithm, we should try to develop a multi-algorithmic composite solver. The idea of multi-algorithms has been explored earlier in conjunction with a multi-processor implementation [1]; the multi-algorithm comprises several algorithms that are simultaneously applied to the problem by exploiting parallelism. We provide a new combinatorial formulation that can be used on uniprocessors and potentially generalized to multiprocessor implementations.

In our model, the composite solver comprises a sequence of different algorithms, thus endowing the composite with the higher cumulative reliability over all member algorithms. Algorithms in the sequence are executed on a given problem until it is solved successfully; partial results from an unsuccessful algorithm are not reused. We provide a combinatorial formulation of the problem in Section 2. Section 3 contains our main contribution in the form of analytical results for obtaining the optimal composite. We provide empirical results on the performance of composite solvers for large sparse linear systems in Section 4 and concluding remarks in Section 5.

## 2 A Combinatorial Model

We now formalize our problem using a combinatorial framework. The composite solver comprises several algorithms and each algorithm can be evaluated on the basis of two metrics: (i) performance or cost, and (ii) reliability. The former can be represented by a normalized value of the performance using either execution time or the number of operations. The reliability is a number in the range  $[0, 1]$  reflecting the probability of successfully solving the problem. For example, if an iterative linear solver fails to converge on average on one fourth of the problems, its failure rate is 0.25 and its reliability is 0.75. In some situations, it may be possible to derive analytic expressions for both metrics. In other situations, these metrics can be computed by empirical means, i.e., by observing the performance of each algorithm on a representative set of sample problems.

Consider generating a composite solver using  $n$  distinct underlying methods (or algorithms)  $M_1, M_2, \dots, M_n$ . Each method  $M_i$ , is associated with its normalized execution time  $t_i$  (performance metric) and reliability  $r_i$ ;  $r_i$  is the success rate of the method and its failure rate is given by  $f_i = 1 - r_i$ . We define the *utility*



**Fig. 1.** Composites of two methods  $M_1, t_1 = 2.0, r_1 = .02$  and  $M_2, t_2 = 3.0, r_2 = .80$ ; both composites have reliability  $.984$  but the composite  $M_2M_1$  has lower execution time.

*ratio* of method  $M_i$  as  $u_i = t_i/ri$ . Let  $\mathbf{P}$  represent the set of all permutations (of length  $n$ ) of  $\{1, 2, \dots, n\}$ . For a specific  $\hat{P} \in \mathbf{P}$ , we denote the associated composite by  $\hat{C}$ .  $\hat{C}$  comprises all the  $n$  underlying methods  $M_1, M_2, \dots, M_n$  in the sequence specified by  $\hat{P}$ . If  $\hat{P}_k$  denotes the  $k$ -th element of  $\hat{P}$ , the composite  $\hat{C}$  consists of methods  $M_{\hat{P}_1}, M_{\hat{P}_2}, \dots, M_{\hat{P}_n}$ . Now for any  $\hat{P} \in \mathbf{P}$ , the total reliability (success percentage) of the composite  $\hat{C}$  is independent of the permutation and invariant at  $1 - \prod_{i=1}^n (1 - r_i)$ , a value higher than that of any component algorithm. Next, observe that  $\hat{T}$ , the worst case execution time of  $\hat{C}$ , is:

$$\hat{T} = t_{\hat{P}_1} + f_{\hat{P}_1} t_{\hat{P}_2} + \dots + f_{\hat{P}_1} f_{\hat{P}_2} \dots f_{\hat{P}_{n-1}} t_{\hat{P}_n}.$$

Thus, the execution times of different composites can indeed vary depending on the actual permutation. A two-method example in Figure 1 shows how the permutation used for the composite affects the execution time. Our goal is to determine the optimal composite, i.e., the composite with minimal worst-case execution time.

We now introduce some additional notation required for the presentation of our analytical results. Consider the subsequence  $\hat{P}_k, \hat{P}_{k+1}, \dots, \hat{P}_l$  ( $\hat{P} \in \mathbf{P}$ ) denoted by  $\hat{P}_{(k:l)}$ . Now  $\hat{P}_{(k:l)}$  can be associated with a composite comprising some  $l - k$  methods using the notation  $\hat{C}_{(k:l)}$ . The total reliability of  $\hat{C}_{(k:l)}$  is denoted by  $\hat{R}_{(k:l)} = 1 - \prod_{i=k}^l f_{\hat{P}_i}$ . Similarly the percentage of failure,  $\hat{F}_{(k:l)} = \prod_{i=k}^l f_{\hat{P}_i}$ . Observe that both these quantities depend only on the underlying set of methods specified by  $\hat{P}_k, \hat{P}_{k+1}, \dots, \hat{P}_l$  and are invariant under all permutations of these methods. We next define  $\hat{T}_{(k:l)}$  as the worst-case time of  $\hat{C}_{(k:l)}$ ; we can see that  $\hat{T}_{(k:l)} = \sum_{i=k}^l [t_{\hat{P}_i} \prod_{m=k}^{i-1} f_{\hat{P}_m}]$ . A final term we introduce is the total utility ratio of  $\hat{C}_{(k:l)}$  denoted by  $\hat{U}_{(k:l)} = \hat{T}_{(k:l)} / \hat{R}_{(k:l)}$ .

For ease of notation, we will drop explicit reference to  $\hat{P}_i$  in expressions for  $\hat{R}, \hat{F}, \hat{T}$ , and  $\hat{U}$  for a specific  $\hat{C}$  and  $\hat{P}$ . Now the expression for  $\hat{T}_{(k:l)}$  simplifies to  $\sum_{i=k}^l [t_i \prod_{m=k}^{i-1} f_m]$ . Additionally, in an attempt to make the notation consistent,

we will treat  $\hat{C}_{(k:k)}$  specified by  $\hat{P}_{(k:k)}$  as a (trivial) composite of one method and use related expressions such as  $\hat{T}_{(k:k)}$ ,  $\hat{R}_{(k:k)}$ ,  $\hat{F}_{(k:k)}$ ,  $\hat{U}_{(k:k)}$  (where  $t_k = \hat{T}_{(k:k)}$ ,  $r_k = \hat{R}_{(k:k)}$ ,  $f_k = \hat{F}_{(k:k)}$ , and  $u_k = \hat{U}_{(k:k)}$ ).

### 3 Analytical Results

This section contains our main analytical results aimed at constructing an optimal composite. Some natural composites include sequencing underlying methods in (i) increasing order of time, or (ii) decreasing order of reliability. Our results indicate that both these strategies are non-optimal. We show in Theorems 1 and 2 that a composite is optimal if and only if its underlying methods are in increasing order of the utility ratio.

We begin by observing that for any  $\hat{P} \in \mathbf{P}$  the composite  $\hat{C}$ , can be viewed as being formed by the sequential execution of two composites,  $\hat{C}_{(1:r)}$  and  $\hat{C}_{(r+1:n)}$ . We can also easily verify that  $\hat{T}_{(1:n)} = \hat{T}_{(1:r)} + \hat{F}_{(1:r)}\hat{T}_{(r+1:n)}$ . We use this observation to show that for any composite, the overall utility ratio is bounded above by the largest utility ratio over all underlying methods.

**Lemma 1.** *For any  $\hat{P} \in \mathbf{P}$ , the utility ratio of the composite  $\hat{C}$  satisfies  $\hat{U} \leq \max\{\hat{U}_{(i:i)} : 1 \leq i \leq n\}$ .*

*Proof.* We can verify (with some algebraic manipulation) that the statement is true for the base case with two methods ( $n = 2$ ). For the inductive hypothesis, assume that the statement is true for any composite of  $n - 1$  methods, that is,  $\hat{U}_{(1:n-1)} \leq \max\{\hat{U}_{(i:i)} : 1 \leq i \leq n - 1\}$ . Now consider  $\hat{C}$ , a composite of  $n$  methods with  $\hat{P}$  as the associated sequence. By our earlier observation, we can view it as a composite of *two methods* with execution times  $\hat{T}_{(1:n-1)}$  and  $\hat{T}_{(n:n)}$ , reliabilities  $\hat{R}_{(1:n-1)}$  and  $\hat{R}_{(n:n)}$ , and utility ratios  $\hat{U}_{(1:n-1)}$  and  $\hat{U}_{(n:n)}$ . If  $\hat{U}_{(1:n-1)} \leq \hat{U}_{(n:n)}$ , then by the base case,  $\hat{U}_{(1:n)} \leq \hat{U}_{(n:n)}$  and by the induction hypothesis,  $\hat{U}_{(1:n)} \leq \max\{\hat{U}_{(i:i)} : 1 \leq i \leq n\}$ . It is also easy to verify that the statement is true if  $\hat{U}_{(n:n)} \leq \hat{U}_{(1:n-1)}$ .  $\square$

**Theorem 1.** *Let  $\tilde{C}$  be the composite given by the sequence  $\tilde{P} \in \mathbf{P}$ . If  $\tilde{U}_{(1:1)} \leq \tilde{U}_{(2:2)} \leq \dots \leq \tilde{U}_{(n:n)}$ , then  $\tilde{C}$  is the optimal composite, i.e.,  $\tilde{T} = \min\{\hat{T} : \hat{P} \in \mathbf{P}\}$ .*

*Proof.* It is easy to verify that the statement is indeed true for the base case for composites of two methods ( $n = 2$ ). We next assume that the statement is true for composites of  $n - 1$  methods. Now we extend the optimal composite of  $n - 1$  methods to include the last method; let this sequence be given by  $\tilde{P}$  and the composite by  $\tilde{C}$ . For the sake of contradiction, let there be a permutation  $\hat{P} \in \mathbf{P}$ , such that  $\hat{T} \leq \tilde{T}$  and the utility ratios  $\{\hat{U}_{(i:i)} : 1 \leq i \leq n\}$  are not in increasing order of magnitude.

Let the  $k$ -th method in  $\hat{C}$  be the  $n$ -th method in  $\tilde{C}$ . Therefore  $\hat{T}_{(k:k)} = \tilde{T}_{(n:n)}$

and  $\dot{F}_{(k:k)} = \tilde{F}_{(n:n)}$ . Using the earlier observations:

$$\tilde{T} = \tilde{T}_{(1:k)} + \tilde{F}_{(1:k)} \tilde{T}_{(k+1:n-1)} + \tilde{F}_{(1:k)} \tilde{F}_{(k+1:n-1)} \tilde{T}_{(n:n)} \quad (1)$$

$$\begin{aligned} \dot{T} &= \dot{T}_{(1:k-1)} + \dot{F}_{(1:k-1)} \dot{T}_{(k:k)} + \dot{F}_{(1:k-1)} \dot{F}_{(k:k)} \dot{T}_{(k+1:n)} \\ &= \dot{T}_{(1:k-1)} + \dot{F}_{(1:k-1)} \tilde{T}_{(n:n)} + \dot{F}_{(1:k-1)} \tilde{F}_{(n:n)} \dot{T}_{(k+1:n)} \end{aligned} \quad (2)$$

We know that  $\tilde{T}_{(1:n-1)}$  is the optimal time over all composites of  $n-1$  methods and thus lower than the time for composite obtained by excluding the  $k$ -th method in  $\dot{C}$  and the  $n$ -th method in  $\tilde{C}$ . Thus  $\dot{T}_{(1:k-1)} + \dot{F}_{(1:k-1)} \dot{T}_{(k+1:n)} \geq \tilde{T}_{(1:k)} + \tilde{F}_{(1:k)} \tilde{T}_{(k+1:n-1)}$ , to yield:

$$\dot{T}_{(1:k-1)} + \dot{F}_{(1:k-1)} \dot{T}_{(k+1:n)} - \tilde{T}_{(1:k)} - \tilde{F}_{(1:k)} \tilde{T}_{(k+1:n-1)} \geq 0 \quad (3)$$

According to our assumption  $\dot{T} \leq \tilde{T}$ ; we expand this relation using Equations 1 and 2 to show that  $\dot{T}_{(1:k-1)} + \dot{F}_{(1:k-1)} \tilde{T}_{(n:n)} + \dot{F}_{(1:k-1)} (1 - \tilde{R}_{(n:n)}) \dot{T}_{(k+1:n)}$  is less than or equal to  $\tilde{T}_{(1:k)} + \tilde{F}_{(1:k)} \tilde{T}_{(k+1:n-1)} + \tilde{F}_{(1:k)} \tilde{F}_{(k+1:n-1)} \tilde{T}_{(n:n)}$ . We can then rearrange the terms on either side to show that the left-hand side of Equation 3 is less than or equal to  $\tilde{F}_{(1:k)} \tilde{F}_{(k+1:n-1)} \tilde{T}_{(n:n)} - \dot{F}_{(1:k-1)} \tilde{T}_{(n:n)} + \tilde{F}_{(1:k-1)} \tilde{R}_{(n:n)} \dot{T}_{(k+1:n)}$ . Thus,

$$0 < \tilde{F}_{(1:k)} \tilde{F}_{(k+1:n-1)} \tilde{T}_{(n:n)} - \dot{F}_{(1:k-1)} \tilde{T}_{(n:n)} + \tilde{F}_{(1:k-1)} \tilde{R}_{(n:n)} \dot{T}_{(k+1:n)}.$$

By rearranging terms and using the equation  $\tilde{F}_{(1:k)} \tilde{F}_{(k+1:n-1)} = \dot{F}_{(1:k-1)} \dot{F}_{(k+1:n)}$  to simplify, we obtain:

$$\begin{aligned} \dot{F}_{(1:k-1)} \tilde{T}_{(n:n)} - \tilde{F}_{(1:k)} \tilde{F}_{(k+1:n-1)} \tilde{T}_{(n:n)} &\leq \dot{F}_{(1:k-1)} \tilde{R}_{(n:n)} \dot{T}_{(k+1:n)}. \\ \dot{F}_{(1:k-1)} \tilde{T}_{(n:n)} - \dot{F}_{(1:k-1)} \dot{F}_{(k+1:n)} \tilde{T}_{(n:n)} &\leq \dot{F}_{(1:k-1)} \tilde{R}_{(n:n)} \dot{T}_{(k+1:n)}. \end{aligned}$$

Canceling the common terms on either side yields  $\tilde{T}_{(n:n)} (1 - \dot{F}_{(k+1:n)}) \leq \tilde{R}_{(n:n)} \dot{T}_{(k+1:n)}$ . Observe that this is equivalent to  $\tilde{U}_{(n:n)} \leq \dot{U}_{(k+1:n)}$ . By the definition of  $\tilde{C}$ ,  $\tilde{U}_{(n:n)}$  is the largest utility ratio among all the  $n$  methods. But if  $\tilde{U}_{(n:n)} \leq \dot{U}_{(k+1:n)}$ , there is a composite whose overall utility is higher than the maximum utility ratio of its component methods, thus contradicting Lemma 1. This contradiction occurred because our assumption that  $\dot{T} \leq \tilde{T}$  is not true; hence the proof.  $\square$

We next show that if a composite is optimal, then its component methods are in increasing order of the utility ratio. The proof uses shortest paths in an appropriately weighted graph.

**Theorem 2.** *If  $\tilde{C}_{(1:n)}$  is the optimal composite then the utility ratios are arranged in increasing order, i.e.,  $\tilde{U}_{(1:1)} \leq \tilde{U}_{(2:2)} \leq \dots \leq \tilde{U}_{(n-1:n-1)} \leq \tilde{U}_{(n:n)}$ .*

*Proof.* Consider a graph constructed with unit vertex weights and positive edge weights as follows. The vertices are arranged in levels with edges connecting

vertices from one level to the next. There are a total of  $n + 1$  levels numbered 0 through  $n$ . Each vertex at level  $l$  ( $0 \leq l \leq n$ ) denotes a subset of  $l$  methods out of  $n$  methods. Assume that the vertex is labeled by the set it represents. Directed edges connect a vertex  $V_S$  at level  $l$  to a vertex  $V_{\bar{S}}$  only if  $|\bar{S} \setminus S| = 1$  and  $\bar{S} \cap S = S$ , i.e., the set  $\bar{S}$  has exactly one more element than  $S$ . Let  $F_S$  denote the total failure rate over all methods in the set  $S$ . If  $\bar{S} \setminus S = \{i\}$ , the edge  $V_S \rightarrow V_{\bar{S}}$  is weighted by  $F_S T_{(i:i)}$ , the time to execute method  $i$  after failing at all previous methods. It is easy to verify that any path from  $V_0$  (representing the empty set) to  $V_{\{1,2,\dots,n\}}$  represents a particular composite, one in which methods are selected in the order in which they were added to sets at subsequent levels. Now the shortest path represents the optimal composite.

Assume we have constructed the shortest path in the graph. Consider a fragment of the graph, as shown in Figure 2. We assume that  $V_S$  is a node on the shortest path, and  $V_{\hat{S}}$  is also a node on the shortest path, such that  $\hat{S} - S = \{i, j\}$ . There will be only 2 paths from  $V_S$  to  $V_{\hat{S}}$ , one including the node  $V_{\bar{S}}$  ( $\bar{S} - S = \{i\}$ ) and the other including the node  $V_{S^*}$  ( $S^* - S = \{j\}$ ). Without loss of generality, assume  $V_{S^*}$  is the node on the shortest path; thus method  $j$  was selected before method  $i$  in the sequence. Let the time from  $V_0$  to  $V_S$  be denoted by  $T_S$  and the failure rate by  $F_S$ . Using the optimality property of the shortest path:

$$T_S + F_S T_{(j:j)} + F_S F_{(j:j)} T_{(i:i)} \leq T_S + F_S T_{(i:i)} + F_S F_{(i:i)} T_{(j:j)}$$

After canceling common terms we get  $T_{(j:j)} + F_{(j:j)} T_{(i:i)} \leq T_{(i:i)} + F_{(i:i)} T_{(j:j)}$ . This can be simplified further using the relation  $F_{(j:j)} = 1 - R_{(j:j)}$  to yield:  $R_{(j:j)} T_{(i:i)} \geq R_{(i:i)} T_{(j:j)}$  and thus  $U_{(j:j)} \leq U_{(i:i)}$ . This relationship between utility ratios holds for any two consecutive vertices on the shortest path. Hence, the optimal composite given by the shortest path is one in which methods are selected in increasing order of the utility ratio.  $\square$

## 4 Empirical Results

Our experiments concern composite solvers for large sparse linear systems of equations. We use a suite of nine preconditioned Conjugate Gradient methods labeled  $M_1, \dots, M_9$ .  $M_1$  denotes applying Conjugate Gradients without any preconditioner.  $M_2$  and  $M_3$  use Jacobi and SOR preconditioning schemes respectively. Methods  $M_4$  through  $M_7$  use incomplete Cholesky preconditioners with 0,1,2 and 3 levels of fill. Methods  $M_8$  and  $M_9$  use incomplete Cholesky preconditioners with numerical drop threshold factors of .0001 and .01.

For our first experiment we used a set of six `bcsstk` sparse matrices from finite element methods in structural mechanics. We normalized the running time of each method by dividing it by the time required for a sparse direct solver. The geometric mean of the normalized running time was used as our estimate of  $t_i$  for each  $M_i$ . We assumed that the method was unsuccessful if it failed to converge in 200 iterations. We used the success rate as the reliability metric  $r_i$  for method

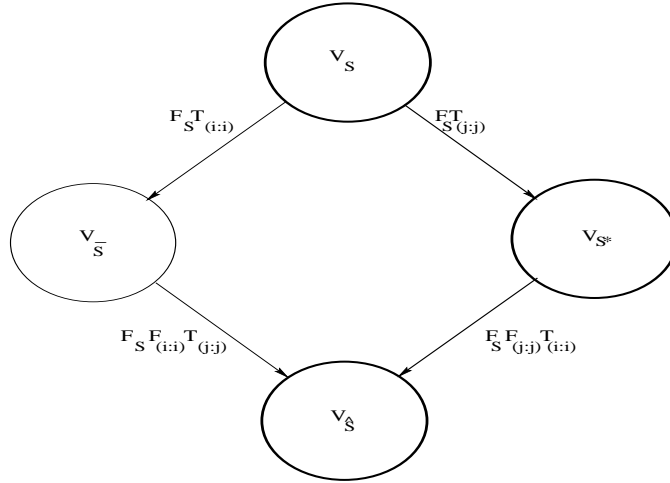


Fig. 2. Segment of the graph used in the proof of Theorem 2.

$M_i$ . These two measures were used to compute the utility ratio  $u_i = t_i/r_i$  for each method  $M_i$ . We created four different composite solvers  $C_T, C_R, C_X, C_O$ . In  $C_T$  underlying methods are arranged in increasing order of execution time. In  $C_R$  the underlying methods are in decreasing order of reliability. The composite  $C_X$  is based on a randomly generated sequence of the underlying methods. The composite  $C_O$  is based on the analytical results of the last section; underlying methods are in increasing order of the utility ratio. The overall reliability of each composite is .9989, a value significantly higher than the average reliability of the underlying methods. We applied these four composite solvers to the complete set of matrices and calculated the total time for each composite over all the test problems. The results are shown in Table 1; our optimal composite  $C_O$  has the least total time.

In our second experiment, we considered a larger suite of test problems consisting of matrices from five different applications. To obtain values of the performance metrics we used a sample set of 10 matrices consisting of two matrices from each application type. We constructed four composite solvers  $C_T, C_R, C_X, C_O$  as in our first experiment. Results in Table 2 indicate that our composite solver still has the least total execution time over all problems in the test suite. The total execution time of  $C_O$  is less than half the execution time for  $C_T$ , the composite obtained by selecting underlying methods in increasing order of time.

These preliminary results are indeed encouraging. However, we would like to observe that to obtain a statistically meaningful result it is important to use much larger sets of matrices. Another issue concerns normalization; we normalized by the time for a sparse direct solver but other measures such as the mean or median of observed times could also be used. These statistical aspects merit further study.

**Table 1.** Results for the `bcsstk` test suite.

Methods and metrics									
	$M_1$	$M_2$	$M_3$	$M_4$	$M_5$	$M_6$	$M_7$	$M_8$	$M_9$
Time	1.01	.74	.94	.16	1.47	2.15	3.59	5.11	2.14
Reliability	.25	.50	.75	.25	.50	.50	.75	1.00	.25
Ratio	4.04	1.48	1.25	.63	2.94	4.30	4.79	5.11	8.56
Composite solver sequences									
$C_T$	$M_4$	$M_2$	$M_3$	$M_1$	$M_5$	$M_9$	$M_6$	$M_7$	$M_8$
$C_R$	$M_8$	$M_3$	$M_7$	$M_2$	$M_5$	$M_6$	$M_1$	$M_4$	$M_9$
$C_X$	$M_9$	$M_8$	$M_1$	$M_5$	$M_3$	$M_2$	$M_7$	$M_6$	$M_4$
$C_O$	$M_4$	$M_3$	$M_2$	$M_5$	$M_1$	$M_6$	$M_7$	$M_8$	$M_9$
Execution time (in seconds)									
Problem	Rank	Non-zeroes ( $10^3$ )	$C_T$	$C_R$	$C_X$	$C_O$			
bcsstk14	1,806	63.4	<b>.25</b>	.98	1.19	.27			
bcsstk15	3,908	117.8	1.88	5.38	9.45	<b>1.22</b>			
bcsstk16	4,884	290.3	1.05	6.60	2.09	<b>.98</b>			
bcsstk17	10,974	428.6	57.40	<b>12.84</b>	16.66	37.40			
bcsstk18	11,948	149.1	4.81	5.70	12.40	<b>2.80</b>			
bcsstk25	15,439	252.2	1.60	21.93	36.85	1.59			
Total execution time			66.99	53.43	78.64	<b>44.26</b>			

## 5 Conclusion

We formulated a combinatorial framework for developing multi-method composite solvers for basic problems in scientific computing. We show that an optimal composite solver can be obtained by ordering underlying methods in increasing order of the utility ratio (the ratio of the execution time and the success rate). This framework is especially relevant with the emerging trend towards using component software to generate multi-method solutions for computational science and engineering applications [2]. Our results can be extended to develop interesting variants; for example, an optimal composite with reliability greater than a user specified value, using only a small subset of a larger set of algorithms. Such “subset composites” can effectively reflect application specific tradeoffs between performance and robustness. Another potential extension could include a model where partial results from an unsuccessful method can be reused in a later method in the sequence.

**Table 2.** Results for the test suite with matrices from five applications.

Methods and metrics									
	$M_1$	$M_2$	$M_3$	$M_4$	$M_5$	$M_6$	$M_7$	$M_8$	$M_9$
Time	.77	.73	.81	.20	1.07	1.48	2.10	.98	.76
Reliability	.50	.60	.90	.50	.70	.60	.60	1.00	.40
Ratio	1.54	1.23	.90	.40	1.53	2.47	3.50	.98	1.91
Composite solver sequences									
$C_T$	$M_4$	$M_2$	$M_9$	$M_1$	$M_3$	$M_8$	$M_5$	$M_6$	$M_7$
$C_R$	$M_8$	$M_3$	$M_5$	$M_2$	$M_6$	$M_7$	$M_1$	$M_4$	$M_9$
$C_X$	$M_9$	$M_8$	$M_7$	$M_6$	$M_5$	$M_3$	$M_2$	$M_1$	$M_4$
$C_O$	$M_4$	$M_3$	$M_8$	$M_2$	$M_5$	$M_1$	$M_9$	$M_6$	$M_7$
Execution time (in seconds)									
Problem	Rank	Non-zeroes ( $10^3$ )	$C_T$	$C_R$	$C_X$	$C_O$			
bcsstk14	1,806	63.4	<b>.31</b>	1.06	1.18	.37			
bcsstk16	4,884	290.3	<b>.97</b>	6.35	2.07	.99			
bcsstk17	10,974	428.6	35.7	<b>13.3</b>	16.3	23.4			
bcsstk25	15,439	252.2	1.61	22.8	36.8	<b>1.60</b>			
bcsstk38	8032	355.5	35.8	33.5	51.5	<b>2.39</b>			
crystk01	4875	315.9	<b>.44</b>	4.03	.84	.47			
crystk03	246,96	1751.1	<b>2.55</b>	35.8	5.45	2.56			
crystm02	139,65	322.90	.32	.40	5.38	<b>.32</b>			
crystm03	246,96	583.77	.60	.72	.73	<b>.60</b>			
msc00726	726	34.52	.13	1.39	.23	<b>.13</b>			
msc01050	1050	29.15	.80	<b>.10</b>	.23	.27			
msc01440	1440	46.27	2.91	.79	2.39	<b>.5</b>			
msc04515	4515	97.70	10.5	<b>1.95</b>	6.10	4.45			
msc10848	10848	1229.77	75.6	101	163	<b>26.3</b>			
nasa1824	1824	39.21	2.46	<b>1.15</b>	1.3	1.80			
nasa2146	2146	72.25	<b>.09</b>	.64	2.29	.09			
nasa2910	2910	174.29	10.9	<b>2.34</b>	6.69	2.80			
nasa4704	4704	104.756	13.4	13.4	13.40	<b>4.61</b>			
xerox2c1	6000	148.05	.27	1.92	<b>.23</b>	18.1			
xerox2c2	6000	148.30	<b>.24</b>	.41	.48	.25			
xerox2c3	6000	147.98	.27	.41	<b>.21</b>	.24			
xerox2c4	6000	148.10	.23	.40	<b>.22</b>	.23			
xerox2c5	6000	148.62	.25	.42	.24	<b>.23</b>			
xerox2c6	6000	148.75	.29	.62	.90	<b>.23</b>			
Total execution time			196.64	244.9	318.16	<b>90.6</b>			

## References

1. Barrett, R., Berry, M., Dongarra, J., Eijkhout, V., Romine, C.: Algorithmic Bombardment for the Iterative Solution of Linear Systems: A PolyIterative Approach. *Journal of Computational and applied Mathematics*, **74**, (1996) 91-110
2. Bramley, R., Gannon, D., Stuckey, T., Villacis, J., Balasubramanian, J., Akman, E., Berg, F., Diwan, S., Govindaraju, M.: Component Architectures for Distributed Scientific Problem Solving. To appear in a special issue of *IEEE Computational Science and Eng.*, 2001
3. Golub, G.H., Van Loan, C.F.: *Matrix Computations* (3rd Edition). The John Hopkins University Press, Baltimore Maryland (1996)