2009 IEEE International Advance Computing Conference (IACC 2009)
Patiala, India, 6–7 March 2009

Choosing Best Algorithm Design Strategy For A Particular Problem

Shailendra Nigam, Dr. Deepak Garg
Department of Computer Science and Engineering Thapar University Patiala

Abstract
This paper proposes choosing the best algorithm design strategy for a particular problem to facilitate the development of best algorithms based upon algorithm design strategy techniques. It describes how a particular algorithm is used for a specific problem. Taking various parameters does a comparison of various algorithms. This paper advocates a wider use of different problems in teaching the best algorithm design strategy.

1. Introduction
Our Studies of how people design algorithms reveal that principal design methods are used based on some parameters in the absence of specific knowledge and we believe that these parameters will play an equally important role in the design of algorithms. A study of algorithms has come to be recognized as the cornerstone of computer science. The progress in this field to date, however, has been very uneven. While the framework for analysis of algorithms has been firmly established and successfully developed for quite some time, much less effort has been devoted to algorithm design techniques. This comparative lack of interest is surprising and unfortunate in view of the two important payoffs in the study of algorithm design techniques: "First, it leads to an organized way to devise algorithms. Algorithm design techniques give guidance and direction on how to create a new algorithm. Though there are literally thousands of algorithms, there are very few design techniques. Second, the study of these techniques help us to categorize or organize the algorithms. Although some algorithms design strategies are better than others on average, there is rarely a best algorithm design strategies for a given problem. Instead, it is often the case that different algorithms design strategies perform well on different problem instances. Not surprisingly, this phenomenon is most pronounced among algorithms for solving hard problems, because runtimes for these algorithms design strategy are often highly variable from instance to instance. Choosing best algorithm design strategy is one of the most difficult decisions. Algorithm design is a specific method to create a mathematical process in solving problems. Applied algorithm design is algorithm engineering. Techniques for designing and implementing algorithm designs are algorithm design patterns , such as template method patterns, decorator patterns, uses of data structures, and name and sort lists. Some current uses of algorithm design can be found in internet retrieval processes of web crawling packet routing and caching. Use employ evolving design strategy to make algorithm. The initial design is a correct, solution to the problem but may not be highly efficient each subsequent design is an improvement or optimization of the prior design, and the final design is an optimal, algorithm for solving the problem. At each stage optimized the strategy is applied and the effect on algorithmic complexity is derived. As each transformation is considered, additional abstractions necessary to express the design strategies are introduced.
The pedagogical advantages of the successive design strategy include:

- Students see design principles applied in a precise context.
- A succession of different principles is applied in the stages of the design for a single problem. This models the design process in the real world.
- At each stage, the demonstration of correctness requires showing only correctness relative to the preceding algorithm. [1,3]

2. Comparison of algorithm Design Strategy
We have compared the various algorithm design strategies on the basis of various factors like complexity, memory required, stability etc. This is very important to know about what is complexity of algorithm in term of time and space. If we blindly use sorting without considering complexity of algorithm then it would be inefficient. Efficiency covers lots of points, including. [1,5]

• CPU (time) usage
• Memory usage
• Disk usage
• Network usage

Be careful to differentiate between, Performances: how much time/memory/disk/... is actually used when a program is run. This depends on the machine, compiler, etc. as well as the code. Complexity, how do the resource requirements of a program or algorithm scale, i.e., what happens, as the size of the problem being solved gets larger? Complexity affects performance but not the other way around. The time required by a method is proportional to the number of "basic operations" that it performs. [3,5,7].

The design strategy is based on the theoretical concept of the algorithm design strategy.

Design Strategy	Advantage	Disadvantage
Brute force	Simple and	Does not uses

	easy to implement	any tactics or short cut
Greedy algorithm	Very large Number of feasible solutions	Does not always gives the optimal solution.
Divide and conquer	Algorithm efficiency	Recursion overhead
Dynamic Programming	Recalls performed calculation	Only for overlapping sub problems
Backtracking Algorithm	1.quick test 2.Pair matching 3. Following real life concept	1.Not widely implemented. 2. Cannot express left-recursive rules 3. More time & complexity
Branch and bound	Very large Number of feasible solutions	Finding pruning strategies require clever thinking technologies

Table 1(Comparison of Design Strategy)

3. Best Algorithm Design Selection for a particular problem

It has long been understood that algorithm performance can vary substantially across different classes of problems. First to formalize algorithm selection as a computational problem, framing it in terms of function approximation. Broadly, algorithm selection can be seen as falling into his framework if identifying the goal of selecting a mapping S (x) from the space of instances to the space of algorithms, to maximize some performance measure perf(S(x), x). We explain our choice of methodology by relating it to other approaches for algorithm design selection that have been proposed in the literature. [3,8]

Some Algorithm Design Strategies

Design Strategies	Based on.
Recursive Algorithm	Reapplying algorithm to sub problem
Backtracking Algorithm	Depth-first recursive search
Divide and Conquer	Dividing problem into subproblems
Dynamic Programming Algorithm	Remembering past results
Greedy Algorithm	Trying best current (local) choice
Brute Force Algorithm	Trying all possible solutions
Branch and Bound Algorithm	Limiting search using current solution
Heuristic Algorithm	Trying to guide search for solution
Decrease-and-Conquer	Extend solution of smaller instance to obtain solution to original instance

Table 2 Algorithm design Information table

Choosing the best Algorithm Design Strategies depends upon the following features:

- Properties of problem
- Expected problem size
- Available resources

Table3 shows the type of problem and which is the best algorithm design for a particular problem. it describes the properties of the problem, expected size, available resources and parameters of the problem and also describes the best algorithm for a particular problem.
A brute-force algorithm would simply multiply a by itself n-l times. A divide-and-conquer algorithm would use the formula $a^n = a^{\lfloor n/2 \rfloor} a^{\lceil n/2 \rceil}$. The decrease by one variety of the decrease and conquer approach yields $a^n=a^{n-1}a$; the decrease by half variety would be based on the formula $a^n = (a^{\lfloor n/2 \rfloor})^2$ for even n's and $(a^{\lfloor n/2 \rfloor})^2 a$ for odd n's. Finally the transformation strategy can be illustrated by two well-known algorithms that exploit the binary representation of n.[6,8,9]

If some guidelines are available regarding the suitability of a particular technique to a problem, then a lot of time can be saved and algorithms may be developed only in that technique method.

Type of Problem	Algorithm Strategies	Example
• Multi-branched recursion. • Hard Problems • Sharing repeated subproblems • Overlapping subproblems • Optimal substructure • Memoization	- Divide-and-conquer algorithms are naturally implemented as recursive procedures. The partial sub-problems leading to the one currently being solved are automatically stored in the procedure call stack. - The dynamic programming algorithm is suitable for the observe the dependency of the sub problem	Fibonacci numbers, Towers of Hanoi, The Halting Problem, geometric curves, Closest-Points Merge sort
• Optimization problems • Heuristic problem • Interval Scheduling	- Brute force Is a straightforward approach to solving a problem, usually directly based on The problem's statement and definitions of the concepts involved. - Greedy algorithms can run significantly faster than brute force ones. Unfortunately, it is not always the case that a greedy strategy leads to the correct solution.	Selection sort, String matching, Convex-hull problem, and Exhaustive search, Traveling salesman problem
• Combinatorial optimization problems	- Backtracking depends on user-given "black box procedures" that define the problem to be solved. - Backtracking is a better approach than brute force (Independently evaluating all possible solutions)	Calculate the path (route)(Example the Traveling Salesman Problem, Minimum Spanning Tree Problem, N Queens, Time and space complexity - Useful when problem size is small - Integer linear programs (ILPs) problems
• Representation problem	Transform and Conquer algorithm basically handle the change one instance to another instance Of the problems so this type of the problem basically suitable for the transform and conquer algorithm.	Heap sort, gaussian elimination, hashing, search trees
• Global optimization problem • Test-Cover Problem	- The branch and bound strategy divides a problem to be solved into a number of subproblems, similar to the strategy backtracking. - Branch and bound algorithm is Sometimes we can tell that a particular branch will not lead to an optimal solution: - The partial solution may already be infeasible - Already have another solution that is guaranteed to be better than any descendant of the given solution	Travelling salesman problem

Table 3. This table shown the types of the problems and define the algorithms and Example of algorithm.[9]

4. Discussion

The objective of the analysis in table3 is that if we come across a new problem then based on the inherent characteristics of the problem, it can be categorized in to particular category and then right algorithm can be written. Using the given strategies we tried this on 100 odd problems taken from different sources and the result was that we are able to figure out exact strategies to be used for 67% of the problems in the first instance. For 9% of the problems, two different strategies were tried to get the efficient algorithm; because the characteristics of these problem does not exactly points to a particular strategy. The remaining 24% could not be categorized into any of the above categories or they were looking similar to multiple categories. So initially it is a good to start and the research will continues further to improve these results so that more problems can be categorized and solved in first instance.

5. Conclusion

Comparison of the various algorithm design strategy are done on the basis of various factors like complexity, memory required, stability etc. After the study of all various algorithm design strategy we find that different algorithms have advantages and disadvantages depending on the different parameters. In all the problems it is seen that many design strategies are tested and algorithm are made using various techniques. If some guidelines are available regarding the suitability of a particular technique to a problem, then a lot of time can be saved and algorithms may be developed only in that technique .

6. References

[1]. Falk H¨Uffner Friedrich-Schiller-Universit¨At Jena Gi-Dagstuhl Research Seminar 06362: Algorithm Engineering September 2006.

[2]. Aho, A.V., Hopcroft, J.E., And Ullman, J.D. The Design And Analysis Of Computer Algorithms. Addison-Wesley, 1974.

[3]. Cormen, Leiserson And Rivest, Introduction To Algorithms, Mcgraw Hill And Mit Press, 1990, 329-333.

[4] Branch And Bound Algorithms - Principles And Examples. Jens Clausen March 12, 1999.

[5]Algorithm Design By Successive Transformation Norman Neff Computer Science Department The College Of New Jersey Trenton, Nj 08650.

[6] N. Lynch. Distributed Algorithms. Morgan Kaufmann, 1996.

[7] Tucker, A. B., Kelemen, C. F., And Bruce, K. B. (2001). Our Curriculum Has Become Math-Phobic Proc. Of The 32th Sigcse Technical Symposium On Computer Science Education (Sigcse01), 243-247.

[8] Sowder, L. And Harel G. (2003). Case Studies Of Mathematics Majors' Proof Understanding, Production, And Appreciation. Canadian Journal Of Science, Mathematics And Technology Education 3(2), Pp. 251-267.

[9] Horowitz, E. Algorithms, Design And Classification Of. In Encyclopedia Of Computer Science, 3-Rd Edition, Ralston, A. And Reilly, E.D., Eds. Van Nostrand Reinhold, 1993, Pp. 33-37. 183.