Algorithm Engineering for solution of some hard Combinatorial Optimization Problems

Synopsis of the proposed research plan

Submitted for the degree of

Doctor of Philosophy

of

Department of Electrical Engineering,
Faculty of Engineering,
Dayalbagh Educational Institute.

By: Sulabh Bansal

Under the Supervision of
Supervisor: Prof. C Patvardhan, DEI.
Co-Supervisor: Prof Anand Srivastav, Kiel.
# Contents

1. **Algorithm Engineering** ......................................................... 4

   1.1. Introduction ............................................................................. 4

   1.2. The Algorithm Engineering Cycle .......................................... 5

   1.3. Algorithm Engineering by Example ..................................... 8

2. **Examples of Combinatorial Problems** .................................. 11

   2.1. Knapsack Problems: ............................................................... 11

       2.1.1. Basic Knapsack problem .................................................. 11

       2.1.2. Bounded knapsack problem .............................................. 11

       2.1.3. Unbounded knapsack problem ......................................... 11

       2.1.4. Multiple-choice knapsack problem .................................. 12

       2.1.5. Subset sum problem ....................................................... 12

       2.1.6. Multiple knapsack problem .............................................. 12

       2.1.7. Multidimensional knapsack problem ............................... 12

       2.1.8. Quadratic knapsack problem: .......................................... 13

   2.2. Graph Problems: ................................................................. 16

       2.2.1. Max-cut ............................................................................. 16

       2.2.2. Minimum Vertex Cover (MVC) ......................................... 17

       2.2.3. Graph Coloring ............................................................... 19

       2.2.4. Min Bi-Section Problem: ................................................ 19

3. **General Purpose Programming on Graphical Processing Unit in CUDA** .......... 20

4. **Quantum-Inspired Evolutionary Algorithms (QEA)** ................. 22

5. **Proposed Work** .................................................................... 27

REFERENCES .................................................................................. 27
Figures and Tables

Figure 1: Gaps Between Theory and Practice in Algorithmics [1] .......................................................... 5
Figure 2: Algorithm Engineering Cycle Originally Proposed [1] ................................................................. 6
Figure 3: Algorithm Engineering Cycle Referred by Chimani and Klein[2] ................................................. 7
Figure 4: Major topics and issues in current algorithm engineering [3] ....................................................... 8
Figure 5: Comparative improvement in performance of CPU and GPU .................................................... 20
Figure 6: Polar plot for rotation gate for Q-bit individuals [5] ............................................................... 24

Table 1: Look table of $\Delta \theta_i$, where $f(.)$ is the profit, and $b_i$ and $x_i$ are the $i$th bits of the best solution $b$ and the binary solution $x$, respectively. In the knapsack problem value $0.01 \pi$ for $\theta_3$, $-0.01 \pi$ for $\theta_5$ and 0 for all others were proposed [5]. ......................................................... 24
1. ALGORITHM ENGINEERING

1.1. Introduction

Engineering is defined as the science, skill and profession of acquiring and applying scientific, economic, social, and practical knowledge in order to design and also build structures, machines, devices, systems, materials and processes.

Engineering as used in naming the disciplines of engineering means development of tools used in the specific discipline. In Computer Engineering, engineering means development of tools for computer systems or solution to the concerned problems.

The term engineering when used with software engineering refers to development, study and analysis of different approaches or methodologies of developing a software solutions or application for the given problems. These methodologies attempt to minimize the time required to develop or modify solution which meets the requirements without compromising the quality of solution. Software engineering methodologies propose the stages for development for software as requirement analysis, design, implementation and testing. Software engineering research keeps on improving the process of developing software.

Algorithm Engineering refers to process of developing algorithms which are practically useful i.e. it includes formulating efficient algorithms to solve a problem, proving their efficiency theoretically focusing on bridging the gap between theory and practice in algorithmics.[1]

"Efforts must be made to ensure that promising algorithms discovered by the theory community are implemented, tested and refined to the point where they can be usefully applied in practice. [...] to increase the impact of theory on key application areas."[2]

Algorithm Engineering gained its importance only recently. In earlier days researchers focused on the theoretical analysis of asymptotic worst-case running times. They used to aim primarily at the development or improvement of algorithms in order to have asymptotically faster algorithms. In this pursuit they developed sophisticated data structures and algorithms. But the facts such that asymptotic analyses might be hiding huge constants or implementing the algorithm and data structures might become practically too complex to be feasible were ignored or were given less importance.

Many times this resulted in algorithms which were theoretically very good and efficient but practically they were either very complex to implement or took unexpectedly huge time. Sometimes theoretically proven good algorithms contained some undetected errors which remained undetected till their first implementation. Chimani and Klein [3] mention several examples of such algorithms. Knuth’s stated as follows:

"Beware of bugs in the above code; I have only proved it correct, not tried it.”[4]

Due to the increasing complexity of applications, algorithms designed to solve the concerning problems give the idea and strategy rather than explaining the nitty-gritty of implementation and data structures to be used.
Figure 1: Gaps Between Theory and Practice in Algorithmics [1]

1.2. The Algorithm Engineering Cycle

Generally research activities concerning Algorithm Engineering (AE) concentrate on the design, the engineering and the theoretical and experimental performance analysis of algorithms. Main research interests deal with the solution of optimization problems and the design of efficient data structures, with special emphasis on applications involving large data sets.

Peter Sanders[1] considers the

- design,
- analysis,
- implementation
- and experimentation

as primary steps of AE. He states that these steps should be executed in cycle giving feedback in order to improve the algorithms in terms of implementation, efficiency .... According to Sanders apart from the above mentioned steps AE stresses on other aspects to ensure the applicability and practical performance of algorithms. These are as follows:

- Using realistic models to describe the problem and machine,
- Using and updating the algorithm libraries to bring in more uniformity and guarantee within the newly written algorithms and
- Evaluating performance of algorithms on realistic and varied benchmark problem set

According to Sanders [1] AE in brief is a “round” methodology for the development of efficient algorithms which simplifies their practical use without abolishing the importance of algorithm theory. Algorithm with proven performance guarantees has a degree of generality, reliability and predictability that cannot be obtained with any number of experiments. Thus AE is a super set which apart from developing and proving the performance according to algorithm theory improve the algorithms in terms of simplicity, constant factors, smoothed analysis etc. He proposed a cycle of AE.
Chimani and Klein[3] elaborate the AE cycle. According to them the cycle usually starts with some specific application in mind (1) and is followed by a search for a realistic model (2) for it such that the solution that is expected matches the requirements as well as possible. The main cycle starts with an initial algorithmic design (3) on how to solve this model. Based on this design the algorithm is analyzed (4) from the theoretical point of view, e.g., to obtain performance guarantees (5) such as asymptotic runtime, approximation ratios, etc. These latter two steps i.e. to find and analyze an algorithm for a given model are essentially the steps traditionally performed by algorithmic theoreticians in the pen-and-paper methodology.

Then algorithmic development is proceeded by implementing (6) the proposed algorithm. The usefulness of this often time-consuming process should never be underestimated. It forms probably the most important step of AE. First of all, one can only succeed in this step if the algorithm is reasonable in its implementation complexity. As noted before, often purely theoretical algorithms have been developed in the past, where it is unclear how to actually transform the theoretical ideas into actual code. Secondly, and in stark contrast to the assumption that theoretical proofs for algorithms are sufficient, it has often been the case that during this step certain flaws and omissions became obvious. There have been papers which take theoretical algorithms and show that the runtime analysis is either wrong, because of some oversight in estimating the complexity of look-ups, or that the algorithmic description has to be extended in order to achieve the suggested running time.

Furthermore, algorithms may assume some static, probably pre-processed, data structure as its input, where certain queries can be performed asymptotically fast. However, in many real-world scenarios such input has to be stored in a more flexible format, e.g., in a linked list instead of an array. When thinking about graphs, this is even more common, as they are often stored as a dynamic graph with adjacency lists, or as compressed or packed adjacency matrices. In such data structures edge look-ups cannot be easily performed in constant time, as for a full adjacency matrix. However, the full unpacked matrix, which may be required by an algorithm to achieve optimal runtime performance, may simply be too large.
to be stored in practice. In such cases, alternative algorithms, which may be asymptotically slower under the assumption of constant-time edge look-ups but require fewer look-ups, may in fact be beneficial.

Such observations can also be made by corresponding experiments (7), which are the final major piece in our cycle. In particular, one should not be interested in toy experiments, but in ones considering real-world data (8). Such input instances have to be at least similar to the problems the algorithm is originally designed for. The benefit of this step is manifold. It allows us to compare different algorithms on a testing ground that is relevant for the practice. One can better estimate the involved constants that have been hidden in the asymptotic runtime analysis, and thereby answer the question of which algorithm – even if asymptotically equivalent – is beneficial in practice. One may also find that certain algorithms may behave counter-intuitively to the theoretical investigation: analytically slower algorithms may be faster than expected, either due to too crudely estimated runtime bounds or because the worst cases virtually never happen in practice. Such situations may not only occur in the context of running time, but are also very common for approximation algorithms, where seemingly weaker algorithms - probably even without any formal approximation guarantee - may find better solutions than more sophisticated algorithms with tight approximation bounds.

Figure 3: Algorithm Engineering Cycle Referred by Chimani and Klein[2]

According to Chimani and Klein [3] a final by-product of the AE cycle should be some kind of algorithm library which facilitates to bridge the gap between academic theory and practitioners in the non academic field.

The primary reasons according to Chimani and Klein [3] for the difference between the theory and practice in algorithmics are:

- The models considered in theory do not exactly reflect the problem instances and the computer architecture.
The Input data often has certain properties which are application specific but theoretical analysis focuses on the worst case scenarios, due to which two algorithms may compare differently in practice than in theory.

The von Neumann model is generally the base of theoretical analysis of algorithms. But modern computer architecture has changed a lot on different aspects like storing programs and data separately, using several layers of memory hierarchy,

There are issues that need to be considered in order to access, analyse and process the huge data sets as a part of AE:

- Memory Efficiency
- Distributed systems and parallelism
- Approximations and Heuristic Algorithms
- Succinct Data Structures

![Figure 4: Major topics and issues in current algorithm engineering [3]](image)

1.3 Algorithm Engineering by Example

A.E. can better be understood by going through the ongoing research on shortest path algorithms. Chimani and Klein [3] have tried to explain the AE using many such examples. The traditional point-to-point (P2P) shortest path problem is defined as given a graph with specified edge lengths, find the shortest edge sequence to connect two given nodes s and t of the graph. A theoretically proved asymptotically good algorithm to solve the above problem is the famous algorithm Dijkstra presented in 1959.

This algorithm's overall running time in theoretical analysis depends on choice of data structures realizing the priority queue used in the Dijkstra's algorithm. Research has long focused on finding more efficient heap data structures to implement priority queue. But in practice really essential bottleneck is the number of edge relaxations and not the complexity of queue operations.

Bi- and Goal Directed Algorithms and
Pruning have been suggested for better performance

Multilevel Approaches are the newest and in conjunction with the above techniques, most successful acceleration techniques.

The highway hierarchies as presented in [5] currently offer the best speed-ups.

Overall, for each such acceleration technique we can construct instances where they are not beneficial. However when applied to real-world instances such as the full road-networks of the USA or Europe, which contain multiple millions of nodes and edges, these techniques allow us to compute provable optimal shortest paths in a couple of microseconds.

Based on the successes for static route finding, current research now also focuses on adapting these algorithms for dynamic settings, taking into account situations where travel times on certain streets are dependent on the current time (e.g., rush hour versus 3 am) or on certain traffic conditions due to road blocks, accidents, etc.

Above example clarifies how AE is useful in developing the algorithms gradually with the objective of perfection.

Sanders and Schulz have worked on balanced graph partitioning and report a scalable parallelization and a number of improvements on the classical multi-level approach which leads to improved partitioning quality [6]. They present a multi-level graph partitioning algorithm which uses local improvement algorithms and global search strategies transferred from multigrid linear solvers. Local improvement algorithms are based on max-flow min-cut computations and more localized FM searches. By combining these techniques, an algorithm that is fast and is able to improve the best known partitioning results for many inputs is presented.

AE with the perspective of experiments is discussed in "A guide to experimental algorithmics" [7].

AE consists of the design, analysis, implementation and experimental evaluation of practicable algorithms. Research in this direction can be done both in a lab environment and in a real world application. Möhring and Berlin explain that there are significant differences in methodology between these two settings [8].

Wagner provides a condensed overview of the techniques enabling the recent development in algorithms for route planning in transportation networks leading to methods that are more than a million times faster than Dijkstra’s algorithm [9]. It is reinstated that, computing shortest paths in huge networks has become a showpiece of AE demonstrating the engineering cycle that consists of design, analysis, implementation and experimental evaluation of practicable algorithms. He discusses new theoretical insights on when and why those techniques work so well.

Protein structural alignment is an important problem in computational biology. Wohlers et al. [10] present first successes on provably optimal pairwise alignment of protein inter-residue distance matrices, using the popular DALI scoring function.
Most often asymptotic results describing only the order of growth are derived. This corresponds to classical analysis of (randomized) algorithms in algorithmics. AE demonstrate that for practical purposes this analysis can be too coarse and more details of the algorithm and its implementation have to be taken into account in order to obtain results that are valid in practice. Jansen & Zarges [11] show that the same holds for evolutionary algorithms. Analyzing the computational complexity of evolutionary algorithms is usually done by analyzing the (expected) optimization time measured by means of the number of function evaluations and describing its growth as a function of a measure for the size of the search space. Using a very recent analysis of a simple evolutionary algorithm Jansen & Zarges demonstrate that counting function evaluations more precisely can lead to results contradicting actual run times. Motivated by these limitations of computational complexity analysis an AE-like approach is presented.

Modern multicore and manycore systems have the strong potential to deliver both high performance and high power efficiency. The large variance in memory access latency, resource sharing, and the heterogeneity of processor architectures in modern multicore and manycore systems raise significant challenges related to AE. Kang et. al. overview important AE issues for modern multicore and manycore systems, and present techniques of AE to address such problems as a guideline for practitioners [12].

Delling et. al. illustrate the use of the scientific method in algorithmics by surveying recent developments in route planning on road networks [13].

Color-coding is a technique to design fixed-parameter algorithms for several NP-complete subgraph isomorphism problems. Hüffner et. al. [14] give various novel algorithmic improvements for color-coding, both from a worst-case perspective as well as under practical considerations.
2. EXAMPLES OF COMBINATORIAL PROBLEMS

Combinatorial problems involve finding a grouping, ordering, or assignment of a discrete, finite set of objects that satisfies given conditions. Candidate solutions are combinations of solution components that may be encountered during a solutions attempt but need not satisfy all given conditions. Solutions are candidate solutions that satisfy all of the given conditions.

2.1. Knapsack Problems:

David Pisinger has dealt with the popular combinatorial problem viz. knapsack problem quite exhaustively and published his work in a form of book [15]. Common to all versions are a set of \( n \) items, with each item \( 1 \leq j \leq n \) having an associated profit \( p_j \) and weight \( w_j \). The objective is to pick some of the items, with maximal total profit, such that the total weight of selected items does not exceed the capacity \( C \). Generally, these coefficients are scaled to become integers, and they are almost always assumed to be positive.

2.1.1. Basic Knapsack problem

The knapsack problem in its most basic form is defined as. The problem is NP-Hard but it can be solved in pseudo polynomial time through dynamic programming[15]:

\[
\text{maximize : } \sum_{j=1}^{n} p_j x_j \\
\text{subject to : } \sum_{j=1}^{n} w_j x_j \leq C \\
x_j \in \{0,1\} \quad \forall j \in \{1, \ldots, n\}
\]

There are several variants of the basic knapsack problem obtained by relaxing some of the constraints.

2.1.2. Bounded knapsack problem

The bounded knapsack problem specifies, for each item \( j \), an upper bound \( u_j \) (which may be a positive integer, or infinity) on the number of times item \( j \) can be selected. The problem is NP-Hard but it can be solved in pseudo polynomial time through dynamic programming[15]. The problem is defined as follows:

\[
\text{maximize : } \sum_{j=1}^{n} p_j x_j \\
\text{subject to : } \sum_{j=1}^{n} w_j x_j \leq C \\
0 \leq x_j \leq u_j, x_j \text{ integral, } \forall j \in \{1, \ldots, n\}
\]

2.1.3. Unbounded knapsack problem

The unbounded knapsack problem (sometimes called the integer knapsack problem) does not put any upper bounds on the number of times an item may be selected. The problem is NP-Hard but it can be solved in pseudo polynomial time through dynamic programming[15]. The problem is defined as follows:

\[
\text{maximize : } \sum_{j=1}^{n} p_j x_j \\
\text{subject to : } \sum_{j=1}^{n} w_j x_j \leq C \\
0 \leq x_j, x_j \text{ integral, } \forall j \in \{1, \ldots, n\}
\]
2.1.4. **Multiple-choice knapsack problem**

If the items are subdivided into \( m \) classes denoted \( N_i \), and exactly one item must be taken from each class, we get the multiple-choice knapsack problem (MCKP). MCKP is NP-Hard but it can be solved in pseudo polynomial time through dynamic programming[15]. It is defined as follows:

maximize : \[ \sum_{i=1}^{m} \sum_{j \in N_i} p_{ij} x_{ij} \]
subject to : \[ \sum_{i=1}^{m} \sum_{j \in N_i} w_{ij} x_{ij} \leq C \]
\[ \sum_{j \in N_i} x_{ij} = 1, \quad i = 1, \ldots, m \]
\[ x_{ij} \in \{0,1\}, \quad i = 1, \ldots, m, \quad j \in N_i \]

2.1.5. **Subset sum problem**

If for each item the profits and weights are identical, we get the subset sum problem (often the corresponding decision problem is given instead). The problem is NP-Hard but it can be solved in pseudo polynomial time through dynamic programming[15]. The problem can defined as follows:

maximize : \[ \sum_{j=1}^{n} p_{j} x_{j} \]
subject to : \[ \sum_{j=1}^{n} p_{j} x_{j} \leq C \]
\[ x_{j} \in \{0,1\} \quad \forall j \in \{1, \ldots, n\} \]

2.1.6. **Multiple knapsack problem**

If we have \( n \) items and \( m \) knapsacks with capacities \( C_i \), we get the multiple knapsack problem. Multiple knapsack problem is strongly NP-complete[16]. The problem can defined as follows:

maximize : \[ \sum_{i=1}^{m} \sum_{j=1}^{n} p_{j} x_{ij} \]
subject to : \[ \sum_{j=1}^{n} w_{ij} x_{ij} \leq C_i \quad \forall i \in \{1, \ldots, m\} \]
\[ \sum_{i=1}^{m} x_{ij} \leq 1 \quad \forall j \in \{1, \ldots, n\}, \forall i \in \{1, \ldots, m\} \]
\[ x_{ij} \in \{0,1\} \quad \forall j \in \{1, \ldots, n\}, \forall i \in \{1, \ldots, m\} \]

2.1.7. **Multidimensional knapsack problem**

The Multidimensional Knapsack Problem (MKP) also known as multi-constrained knapsack problem is a well-studied most general form of knapsack problem. It is strongly NP-hard combinatorial optimization problem occurring in many different applications[17]. When there are \( m \) knapsacks (or resources) with \( i^{th} \) knapsack having \( C_i \) capacity and each item \( j \) when selected occupies \( w_{ij} \) space in \( i^{th} \) knapsack (or consume \( w_{ij} \) amount of \( i^{th} \) resource) problem becomes multidimensional knapsack problem. The MKP can be defined by the following ILP:

maximize : \[ \sum_{j=1}^{n} p_{j} x_{j} \]
subject to : \[ \sum_{j=1}^{n} w_{ij} x_{j} \leq C, \quad i = 1, \ldots, n, \]
\[ x_{j} \in \{0,1\} \quad \forall j \in \{1, \ldots, n\} \]
2.1.8. Quadratic knapsack problem.
Assume there are a set of $n$ items, with each item $1 \leq j \leq n$ having an associated weight $w_j$. In addition we are given $n \times n$ non negative integer profit matrix, $P = (p_{ij})$, where $p_{jj}$ is the profit achieved if item $j$ is selected and $p_{ij} + p_{ji}$ is a profit achieved if both items $i$ and $j$ are selected. The objective is to pick items in such a way that maximum total profit is achieved, such that the total weight of selected items does not exceed the capacity $C$. Quadratic knapsack problem (QKP) is known to be NP-hard in the strong sense\[18\]. The problem can be defined as follows:

$$\text{maximize : } \sum_{i=1}^{n} \sum_{j=1}^{n} p_{ij} x_i x_j$$
$$\text{subject to : } \sum_{j=1}^{n} w_j x_j \leq C$$
$$x_j \in \{0,1\} \quad \forall j \in \{1, \ldots, n\}$$

Guler et. al. discuss a greedy algorithm for knapsack problems and calculated the guarantee value \[19\]. Maximization and minimization versions of knapsack problems and greedy algorithms for these problems are discussed. Guarantee values of the algorithms are calculated and complementary problems are created in order to improve known guarantee values. The guarantee values of the well-known algorithms are calculated as ‘1/2’ for maximization KP and ‘2’ for minimization KP. Using this approach, authors observed that guarantee values of the algorithms are better than others as specified in theorems.

Boyer et. al. present a parallel implementation via CUDA of the dynamic programming method for the knapsack problem on NVIDIA GPU. A GTX 260 card with 192 cores (1.4 GHz) \[20\]. Processing times obtained with the parallel code are compared to the sequential one on a CPU with an Intel Xeon 3.0 GHz. The results show a speedup factor of 26 for large size problems. Furthermore, in order to limit the communication between the CPU and the GPU, a compression technique is presented which decreases significantly the memory occupancy. For the problem of 100,000 items time taken by parallel implementation is reported as around 290 sec where the sequential implementation took more than 7200 sec.

Lalami & El-Baz propose an efficient implementation of the branch and bound method for knapsack problems on a CPU-GPU system via CUDA \[21\]. A better management of GPU memories, less GPU CPU communications and better synchronization between GPU threads are proposed in the implementation in order to increase efficiency. A series of computational results has been displayed and analyzed showing a substantial speedup on a Tesla C2050 GPU. The authors claim that computational results prove that their approach is efficient and has obtained stable speedups of around 20 for difficult knapsack problems. Authors have also claimed that the approach permitted to solve problems with size 500 without exceeding the memory occupancy of the GPUs. For solving the problem of size 500 items it is shown that the serial implementation took 13.39 sec while parallel implementation took 0.65 sec only.

Nowotniak and Kucharski present the parallel implementation of Quantum-Inspired Genetic Algorithm to solve the knapsack problem in a GPU-based massively parallel computing environment (NVidia CUDA™ technology) \[22\]. The paper concerned efficient parameters
tuning (meta-optimization) of the metaheuristic. The authors show when the computations are distributed to eight GPU devices over 400x speedup has been gained. This approach allowed efficient meta-optimization of the algorithm which had been demonstrated on combinatorial optimization (knapsack problem). According to the paper time taken to solve a knapsack problem of 250 items by the serial implementation is 20 to 140 sec for different sizes of populations while time taken by parallel implementation is 2 to 20 sec.

Pospichal et. al. have identified strong and weak points of GPU architecture and proposed parallel genetic algorithm model implemented in CUDA running entirely on the GPU to solve the 0/1 knapsack problem [23]. They show that GPU must be utilized for sufficiently long time in order to obtain reasonable program speedup. Results quality and speed of the model have also been compared with single-threaded CPU code implemented using Galib code. Peak speedup of GPU GA execution performance is 1340x resp. 134x for 4-bit resp. 40-bit problem instances while maintaining reasonable results quality.

Jansen proposes an improved efficient polynomial time approximation scheme (EPTAS) for the multiple knapsack problem [24]. The EPTAS for multiple knapsack problem has running time complexity as \(2^{O(1/(\epsilon^6)\log(1/\epsilon))} \times \text{poly}(n)\) which beat earlier presented running time of \(n^{O(1/(\epsilon^6)\log(1/\epsilon))}\). An improved EPTAS with running time \(2^{O(1/(\epsilon^6)\log(1/\epsilon))} + \text{poly}(n)\) is also reported. The paper also shows that if the integrality gap between the ILP and LP objective values for bin packing with different sizes is bounded by a constant, the running time could be further improved to \(2^{O(1/(\epsilon^2)\log(1/\epsilon))} + \text{poly}(n)\).

Karim & Ryan present an Attribute Grammar with Lookahead (AG+LA) approach [25], a technique to solve heavily constrained Multiple Knapsack Problem. This approach incorporates a form of lookahead into the mapping process of Grammatical Evolution (GE) using Attribute Grammar (AG) to focus only on feasible solutions, thereby avoiding issues such as repeated remapping and introns, both of which are limitations of previous approaches based on AG. Function value has been compared with the optimal. Also compared the algorithm suggested with its' predecessor on the basis of function evaluations and function value. Authors have shown results for the problems of size as big as 105 items and 30 knapsacks.

Lalami et. al. present a heuristic which derived a feasible solution for the Multiple Knapsack Problem [16]. The proposed heuristic called RCH, is a recursive method that performs computation on the core of knapsacks. The RCH heuristic has been compared with the MTHM heuristic of Martello and Toth. Computational results on randomly generated instances have shown that the proposed approach gives better gap and smaller restitution times. The results are reported for problem of size as big as having 100,000 items and 10 knapsacks solved within 600 sec with profit 0.00544\% less than the results obtained by Mortello and Toth [26].

Suri et. al. propose a new GPU-based approach to accelerate the multiple-choice knapsack problem, which is a general version of the 0-1 knapsack problem [27]. Apart from exploiting the parallelism offered by the GPUs, authors also employed a variety of GPU-specific optimizations to further accelerate the running times of the knapsack problem. Moreover, technique is described as scalable in the sense that even when running large instances of
the multiple-choice knapsack problems, authors could efficiently utilize the GPU compute resources and memory bandwidth to achieve significant speedups. Suri et. al. reported 1200 sec for the large problem of size 100 knapsacks and 1024 items in each choice.

Alvesa et. al describe a new solution approach for the multiple choice multidimensional knapsack problem\[28\]. The multiple choice multidimensional knapsack problem is found to be more difficult to solve in part because of its choice constraints. A new hybrid heuristic is proposed that embeds several new procedures for this problem. Authors have reported time taken to be 316 secs for problem having m = 10 dimensions, n = 350 classes and ni = 20 items/class.

Apart from results on synthetic test-cases, Suri demonstrates the applicability of his GPU-based technique in practice by considering a case-study of multiple-choice knapsack problem from system-level design \[29\].

Hyper-heuristics are a class of high-level search techniques which operate on a search space of heuristics rather than directly on a search space of solutions. Earlier hyperheuristics focussed on selecting and applying a low-level heuristic at each stage of a search. Recent trends in hyper-heuristic research have led to a number of approaches being developed to automatically generate new heuristics from a set of heuristic components. Drake et. al. \[30\] investigate the suitability of using genetic programming as a hyper-heuristic methodology to generate constructive heuristics to solve the multidimensional 0-1 knapsack problem. A population of heuristics to rank knapsack items have been trained on a subset of test problems and then applied to unseen instances. The results over a set of standard benchmarks have shown that genetic programming can be used to generate constructive heuristics which yield human-competitive results.

It is well known that the standard (linear) knapsack problem can be solved exactly by dynamic programming in O(n\(c\)) time, where n is the number of items and c is the capacity of the knapsack. The quadratic knapsack problem, on the other hand, is NP-hard in the strong sense, which makes it unlikely that it can be solved in pseudo-polynomial time. Fomeni & Letchford show however that the dynamic programming approach to the linear knapsack problem can be modified to yield a highly effective constructive heuristic for the quadratic version \[18\]. The lower bounds obtained by the heuristic presented consistently within a fraction of a percent of optimal.

Caprara et. al. propose an exact branch-and-bound algorithm for QKP, where upper bounds are computed by considering a Lagrangian relaxation that is solvable through a number of (continuous) knapsack problems \[31\]. Suboptimal Lagrangian multipliers have been derived by using subgradient optimization and provided a convenient reformulation of the problem. Exact solution of instances with up to 400 binary variables are reported. The key point of the improvement was that the upper bounds obtained are typically within 1% of the optimum, but can still be derived effectively. It is also shown that the algorithm is capable of solving reasonable-size Max Clique instances from the literature.

Julstrom presents two genetic algorithms which encode candidate solutions to the QKP problem as permutations of objects \[32\]. One GA applies no heuristic steps while a second
seeds its population and performs crossover by considering the values of objects relative to
the objects already in the knapsack. Both algorithms perform well, and competitively with
two earlier binary-coded GAs. The heuristic measures improve performance significantly.

Kochenberger et. al. provide a comparison of quadratic and linear representations of QKP
based on test problems with multiple knapsack constraints and up to eight hundred
variables [33]. For the linear representations, three standard linearizations are investigated.
Both the quadratic and linear models are solved by standard branch-and-cut optimizers
available via CPLEX. The results show that the linear models perform well on small problem
instances but for larger problems the quadratic model outperformed the linear models tested
both in terms of solution quality and solution time by a wide margin. Moreover, paper
demonstrate that QKP instances larger than those previously addressed in the literature as
well as instances with multiple constraints could be successfully and efficiently solved by
branch and cut methodologies.

2.2. Graph Problems:
Some combinatorial problems in graphs which are of interest are as follows-

2.2.1. Max-cut
The Max-cut problem is one of many NP-complete [34] graph theory problems which
attracted many researchers over the years. Though there is almost no hope in finding a
polynomial time algorithm for max-cut problem, various heuristics, or combination of
optimization and heuristic methods have been developed to solve this problem.

A cut in a weighted undirected graph $G_w = (V,E)$, is defined as partition of the vertices of G
into two sets; and the weight of a cut is the summation of weights of the edges that has an
end point in each set (i.e. the edges that connect vertices of one set to the vertices of the
other). Trivially, one can define the max-cut problem as the problem of finding a cut in G
with maximum weight.

Let us assign a variable $x_i$ to each vertex of $G_w = (V,E)$, and define this variable as follows:

$$ x_i = \begin{cases} 
1 & \text{if the ith vertex is in } S \\
-1 & \text{if the ith vertex is in } \overline{S} 
\end{cases} $$

where $S$ is a subset of $V$, and $\overline{S}$ is the complement of $S$. The max-cut problem is modelled
as:

$$ \text{Max } \frac{1}{4} \sum_{i<j} (1 - x_i x_j) $$

such that: $x_i \in \{+1, -1\} \quad i = 1, ..., n$

Kochenberger et. al. report the application of a new Tabu Search algorithm to large scale
Max-cut test problems [35]. The method provided best known solutions for many well-
known test problems of size up to 10,000 variables, although it has been designed for the
general unconstrained quadratic binary program (UBQP), and has not been specialized in
any way for the Max-Cut problem.
Krisiok et al. present an improved algorithm for finding exact solutions to Max-Cut and the related binary quadratic programming problem, both classic problems of combinatorial optimization [36]. The algorithm used a branch-(and-cut-)and-bound paradigm, using standard valid inequalities and nonstandard semidefinite bounds. Extensive experiments show that algorithm of paper dominated the best existing method.

The Goemans-Williamson algorithm of 1995, that quite substantially contributed to widen the interest for Semi Definite Programming (SDP) based techniques, plays a fundamental role not only for Max-Cut but also for some related problems. Some of those are briefly described in [37] like Max-k-Cut, Coloring and Ordering problems.

Galli et. al. prove several complexity results about the gap inequalities for the max-cut problem [38]. They prove (i) the gap-1 inequalities do not imply the other gap inequalities, unless NP=Co NP; (ii) there must exist non-redundant gap inequalities with exponentially large coefficients, unless NP=Co NP; (iii) the associated separation problem can be solved in finite (doubly exponential) time.

Laurent & Poljak introduced a class of valid inequalities for the max-cut problem, called gap inequalities, which include many other known inequalities as special cases. Galli et. al. present the first ever computational results [39]. In particular heuristic separation algorithms for gap inequalities and their special cases have been described, and it is shown that an LP-based cutting-plane algorithm based on these separation heuristics can yield very good upper bounds in practice.

In[34] the boundary of tractability for the Max-Cut problem in graphs is studied. Main result have shown that Max-Cut above the Edwards-Erdős bound is fixed-parameter tractable: an algorithm is given that for any connected graph with n vertices and m edges finds a cut of size \((m/2)+(n−1)/4+k\) in time \(2^{O(k)} \cdot n^4\), or decides that no such cut exists. Thus a longstanding open question is answered from parameterized complexity that has been posed a number of times over the past 15 years.

### 2.2.2. Minimum Vertex Cover (MVC)

A simple graph \(G\) with \(n\) vertices consists of a set of vertices \(V\), with \(|V| = n\), and a set of edges \(E\), such that each edge is an unordered pair of distinct vertices. A vertex cover \(C\) of \(G\) is a set of vertices such that for every edge \(\{u, v\}\) of \(G\) at least one of \(u\) or \(v\) is in \(C\). Given a vertex cover \(C\) of \(G\) and a vertex \(v\) in \(C\), we say that \(v\) is removable if the set \(C - \{v\}\) is still a vertex cover of \(G\). Denote by \(\rho(C)\) the number of removable vertices of a vertex cover \(C\) of \(G\). A minimal vertex cover has no removable vertices. A minimum vertex cover is a vertex cover with the least number of vertices. Note that a minimum vertex cover is always minimal but not necessarily vice versa. A minimum vertex cover problem is one of the most classical NP-complete problems[40].

Onak et. al. gave a nearly optimal sublinear-time algorithm for approximating the size of a MVC in a graph \(G\) [41]. The algorithm may query the degree \(\text{deg}(v)\) of any vertex \(v\) of its choice, and for each \(1 \leq i \leq \text{deg}(v)\), it may ask for the \(i^{th}\) neighbour of \(v\). Letting \(\text{VC}_{\text{opt}}(G)\) denote the minimum size of vertex cover in \(G\), the algorithm outputs, with high constant success probability, an estimate \(\overline{\text{VC}}(G)\) such that \(\text{VC}_{\text{opt}}(G) \leq \overline{\text{VC}}(G) \leq 2\text{VC}_{\text{opt}}(G) + \epsilon n\),
where $\varepsilon$ was a given additive approximation parameter. The query complexity and running time of the algorithm are $O(\tilde{d} \cdot \text{poly}(1/\varepsilon))$, where $\tilde{d}$ denotes the average vertex degree in the graph. The best previously known sublinear algorithm, had query complexity and running time $O(d^4/\varepsilon^2)$, where $d$ is the maximum degree in the graph. Given the lower bound of $\Omega(\tilde{d})$ (for constant $\varepsilon$) for obtaining such an estimate (with any constant multiplicative factor) the result presented is nearly optimal.

Voß and Fink apply a modified reactive tabu search approach for solving the MVC [42]. Authors propose to replace the random walk by a controlled simulated annealing. Numerical results have been presented outperforming previous metaheuristic approaches in most cases. For graph having number of vertices as 500 and that of edges as 5000 the paper reported time taken ranging from 6 ms to 600 ms for different parameters.

Wei et. al. aiming at detecting the solution space of MVC define a new structure named mutual-determination for MVC on general graphs, which results in the emergence of strong correlations among the unfrozen nodes [43]. Compared with the contemporary algorithms, the presented Mutual-determination and Backbone Evolution Algorithm perform as well as the replica symmetry one in a certain interval but has a small gap higher than the replica symmetric breaking one and has a relatively small error for the exact results. The algorithm with the mutual-determination provides a new viewpoint to solve MVC and understands the organizations of the solution spaces, and the reduced solution graph gives an alternative way to catch detailed information of the ground/steady states.

Cardinal et al. consider the MVC problem in hypergraphs in which every hyperedge has size $k$ (also known as Minimum Hitting Set problem, or minimum set cover with element frequency $k$) [40]. In particular, a randomized polynomial-time algorithm is provided with approximation factor $k/(1+(k-1)\tilde{d}/k\Delta)$, where $\tilde{d}$ and $\Delta$ are the average and maximum degree, and $\Delta$ must be $\Omega(n^{k-1}/\log n)$. The proposed algorithm generalizes a recursive sampling technique for MVC in dense graphs. As a corollary, an approximation factor arbitrarily close to $k/(2−1/k)$ for subdense regular hypergraphs is obtained, which is shown to be the best possible under the Unique Games conjecture.

Cai et al. propose two strategies to design efficient local search algorithms for the MVC problem [44]. Two main drawbacks in state-of-the-art MVC local search algorithms have been observed as:

1. They select a pair of vertices to be exchanged simultaneously, which is time consuming;
2. Although they use edge weighting techniques, they do not have a strategy to decrease the weights.

To address these drawbacks, two new strategies are proposed to design a new algorithm dubbed NuMVC. The experimental results have shown that NuMVC significantly outperform existing state-of-the-art heuristic algorithms on most of the hard DIMACS instances and all instances in the hard random BHOSLIB benchmark. The results are reported for instances as big as $|V|= 4000$, $|E|= 572,774$ and $|V|= 3000$, $|E|= 5,000,000$. Time taken reported for different instances varied from 0.24 sec to 1994 sec.
2.2.3. Graph Coloring
A (vertex) coloring of a graph $G$ is a mapping $c : V(G) \rightarrow S$. The elements of $S$ are called colors; the vertices of one color form a color class. If $|S| = k$, we say that $c$ is a $k$-coloring (often we use $S = \{1, \ldots, k\}$). A coloring is proper if adjacent vertices have different colors. A graph is $k$-colorable if it has a proper $k$-coloring. In a proper coloring, each color class is a stable set. Hence a $k$-coloring may also be seen as a partition of the vertex set of $G$ into $k$ disjoint stable sets $S_i = \{v \mid c(v) = i\}$ for $1 \leq i \leq k$. Therefore $k$-colorable are also called $k$-partite graphs. Moreover, 2-colorable graphs are very often called bipartite. The smallest number of colors needed to color a graph $G$ is called its chromatic number. To find the chromatic number of the graph is called Graph coloring problem which is a well known NP-Complete problem.

2.2.4. Min Bi-Section Problem:
Let $G = (V, E)$ be an undirected graph with node set $V = \{1, \ldots, n\}$ and edge set $E \subseteq \{(i, j) \mid i, j \in V, i < j\}$. For an edge $(i, j)$ for given vertex weights $f_i \in \mathbb{N} \cup \{0\}$, $i \in V$, and edge costs $w_{ij} \in \mathbb{R}$, a partition of the vertex set $V$ into two disjoint clusters $S$ and $V \setminus S$ with sizes $f(S) := \sum_{i \in S} f_i \leq F$ and $f(V \setminus S) \leq F$, where $F \in \mathbb{N} \cap [\frac{1}{2} f(V), f(V)]$, is called a bisection. The minimum bisection problem (MB) asks for a bisection such that the total cost of edges in the cut $\delta(S) := \{i j \in E : i \in S \land j \in V \setminus S\}$ is minimal. The problem is known to be NP-hard[45].

While semidefinite relaxations are known to deliver good approximations for combinatorial optimization problems like graph bisection, their practical scope is mostly associated with small dense instances. For large sparse instances, cutting plane techniques are considered the method of choice. These are also applicable for semidefinite relaxations via the spectral bundle method, which allows to exploit structural properties like sparsity. In order to evaluate the relative strengths of linear and semidefinite approaches for large sparse instances, Arbuster et al. propose a common branch-and-cut framework for linear and semidefinite relaxations of the minimum graph bisection problem [46]. Through extensive numerical experiments it is shown that this semidefinite branch-and-cut approach without problem specific cuts is a superior choice to the classical simplex approach exploiting bisection specific inequalities on a clear majority of our large sparse test instances from VLSI design and numerical optimization.

Mezuman et al. present a framework that seeks and finds the optimal solution of several NP-hard graph partitioning problems [47]. It is shown that in many cases the approach yields the global optimum and improves the popular spectral solution based on experiments. The basic building blocks of the method presented are: (1) Linearizing a ratio problem to get the lambda question (2) Convex message passing algorithm with a bound on the objective function (3) Efficient MAP inference with high order potentials.
3. GENERAL PURPOSE PROGRAMMING ON GRAPHICAL PROCESSING UNIT IN CUDA

Till recent past the computing speed of personal computer was improved by increasing the clock speed of single processor. Because of some fundamental limitations in fabrication of integrated circuits such as power and heat restrictions and approaching physical limit to transistor size researchers and manufacturers introduced multicore CPU’s in personal computers to improve the performance. Over the last few years market has seen the release of up to eight core CPU’s in personal computers. This trend is sometimes referred as multicore revolution.

In 1990’s the demand for speed and performance in applications with 3D graphics escalated. So stress on improving graphical processing increased which gave rise to separate Graphical processing unit in personal computers which were generally installed on the motherboard as an external processor. Initially the GPU’s were restricted to be used only for graphical processing.

At the same time CPU’s reached the limit of serial performance GPU’s were growing exponentially in performance due to massive parallelism. GPU’s come with large number of cores and are comparatively cheaper than CPU. Because of these reasons apart from increasing the cores in CPU’s, researchers were aiming to use the GPU’s as subordinates of CPU to increase the general purpose processing speed of personal computers. In the last half of decade 2000’s the manufacturers introduced GPU’s with an aim to alleviate limitations that prevented them to be used for general purpose computation. In 2006 Nvidia came up with Cuda Architecture with the same intentions.

CUDA (compute unified device architecture) refers to the family of graphics processors which were included with a unified shader pipeline, allowing each and every arithmetic logic unit (ALU) on the chip to be controlled by a program intending to perform general-purpose computations, these ALUs were built to comply with IEEE requirements for single-precision floating-point arithmetic and were designed to use an instruction set tailored for general computation rather than specifically for graphics. Furthermore, the execution units on the GPU were allowed arbitrary read and write access to memory as well as access to a software-managed cache known as shared memory. All of these features of the CUDA Architecture were added in order to create a GPU that would excel at computation in addition to performing well at traditional graphics tasks.

Figure 5: Comparative improvement in performance of CPU and GPU
NVIDIA took industry standard C and added a relatively small number of keywords in order to harness some of the special features of the CUDA Architecture. NVIDIA made public a compiler for this language, CUDA C, to facilitate general-purpose computing on GPUs.

Evolutionary Algorithm, being population based, are inherently parallel and these novel architectures can be exploited to utilize EAs for the solution of more complex problems than hitherto possible.
4. QUANTUM-INSPIRED EVOLUTIONARY ALGORITHMS (QEA)

Quantum mechanical computers were proposed in the early 1980s, these computers were shown to be more powerful than classical computers on various specialized problems. Research on merging evolutionary computing and quantum computing has been started since late 1990s. Han and Kim [48] proposed a novel evolutionary algorithm, called a quantum-inspired evolutionary algorithm (QEA), which is based on the concept and principles of quantum computing such as a quantum bit and superposition of states. Like the EAs, QEA is also characterized by the representation of the individual, the evaluation function, and the population dynamics. However, instead of binary, numeric, or symbolic representation, QEA uses a Q-bit as a probabilistic representation, defined as the smallest unit of information. A Q-bit individual is defined by a string of Q-bits. The Q-bit individual has the advantage that it can represent a linear superposition of states (binary solutions) in search space probabilistically. Thus, the Q-bit representation has a better characteristic of population diversity than other representations. A Q-gate is also defined as a variation operator of QEA to drive the individuals toward better solutions and eventually toward a single state. Initially, QEA can represent diverse individuals probabilistically because a Q-bit individual represents the linear superposition of all possible states with the same probability. As the probability of each Q-bit approaches either 1 or 0 by the Q-gate, the Q-bit individual converges to a single state and the diversity property disappears gradually. By this inherent mechanism, QEA can treat the balance between exploration and exploitation. It should be noted that although QEA is based on the concept of quantum computing, QEA is not a quantum algorithm, but a novel evolutionary algorithm for a classical computer.

The quantum inspired Evolutionary Algorithm given by Hann and Kim [49] is described in brief here.

QEA uses quantum bits (qubits) as the smallest unit of information for representing individuals. Each qubit is represented as $q_i = \begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix}$, where $\alpha_i, \beta_i$ are complex numbers representing probabilistic state of qubit so that $|\alpha_i|^2$ is the probability of state being 1 and $|\beta_i|^2$ is the probability of state being 0 such that $|\alpha_i|^2 + |\beta_i|^2 = 1$. For QEA $\alpha_i, \beta_i$ are considered real without losing the generality. Hann and Kim [49] presented their novel QEA as follows where Q(t) is the qubit population, P(t) is the population of individual solutions, B(t) is the set of best solutions corresponding to each individuals:

```
Procedure QEA
begin
    t ← 0
    initialize Q(t)
    make P(t) by observing the states of Q(t)
    repair P(t)
    evaluate P(t)
    store the best solutions among P(t) into B(t)
    while ( t<MAX_GEN)
    begin
        t ← t+1
```
make $P(t)$ by observing the states of $Q(t-1)$
repair $P(t)$
evaluate $P(t)$
update $Q(t)$
store the best solutions among $B(t-1)$ and $P(t)$ into $B(t)$
if (migration-period)
then migrate $b$ or $b^i$ to $B(t)$ globally or locally, respectively.
end
end

In the initialize step the elements of $Q$ bits, $[\alpha \beta]$, $\alpha, \beta$ were proposed to be initialized to value $1/\sqrt{2}$.
Procedure make is defined as follows.

Procedure Make ($x$)
begin
$i \leftarrow 0$
while ($i < m$) do
begin
$i \leftarrow i + 1$
if (random $[0, 1) < |\beta|^2$)
then $x_i \leftarrow 1$
else $x_i \leftarrow 0$
end
end

The repair function is meant for correcting the solution obtained after collapsing the qubits such that the corrected solution satisfies all of the constraints. It has to be defined as per the requirements of the problem.

Han and Kim[49] mentioned the use of two repair functions (REP1 and REP2) for the purpose of experiments done on the popular combinatorial problem, Knapsack Problem. They determined that REP2, which was named as greedy repair (pseudo code not given), is better than REP1. Greedy Repair used the profit by weight ratio of items as a criterion for selecting an item to delete from knapsack.

Hann and Kim defined a rotational Q-gate used in update procedure for updating the quantum bits used in representation of individuals. It is illustrated by its polar diagram in Figure 6.

For the values of $\Theta$ used in the rotation gate, a lookup table has to be defined as per the requirements of the problem to be solved. A lookup table as was proposed and used by Hann and Kim for solving the knapsack problem is shown in Table 1.
Table 1: Look table of $\Delta \theta_i$, where $f(.)$ is the profit, and $b_i$ and $x_i$ are the $i$th bits of the best solution $b$ and the binary solution $x$, respectively. In the knapsack problem value $0.01\pi$ for $\theta_3$, $-0.01\pi$ for $\theta_5$ and $0$ for all others were proposed [5].

<table>
<thead>
<tr>
<th>$x_i$</th>
<th>$b_i$</th>
<th>$f(x) \geq f(b)$</th>
<th>$\Delta \theta_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>false</td>
<td>$\theta_1$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>true</td>
<td>$\theta_2$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>false</td>
<td>$\theta_3$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>true</td>
<td>$\theta_4$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>false</td>
<td>$\theta_5$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>true</td>
<td>$\theta_6$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>false</td>
<td>$\theta_7$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>true</td>
<td>$\theta_8$</td>
</tr>
</tbody>
</table>

The traditional quantum evolutionary algorithm takes a long time to converge and can be easy trap into local optima. In order to overcome and accelerate the speed of the convergence, a new quantum evolutionary algorithm is proposed by Wang et. al. [50]. The proposed new algorithm named discrete quantum bee colony algorithm incorporate the basic idea of the artificial bee colony algorithm. The initial population is initialized randomly using quantum encode and the population is formed by there parts and every subpopulation evolve cooperatively. In the end, the individual is rated according to the multi-granularity mechanism and also rated according to the evolution condition. Simulation results of knapsack problems show that the proposed algorithm performs better than other algorithm.

To improve the performance of Quantum-inspired Evolutionary algorithm based on P Systems (QEPS), Yarlagadda et. al. present an improved QEPS with a Dynamic Membrane Structure (QEPS-DMS) to solve knapsack problems [51]. QEPS-DMS combines QEA with a P system with a dynamic membrane structure. In QEPS-DMS, a QEA is considered as a sub-algorithm to put inside each elementary membrane of a one-level membrane structure, which is dynamically adjusted in the process of evolution by applying a criterion for measuring population diversity. The dynamic adjustment includes the processes of membrane dissolution and creation. Knapsack problems are applied to test the effectiveness
of QEPS-DMS. Experimental results show that QEPS-DMS outperforms QEPS and three variants of QIEAs recently reported in the literature.

Qin et. al. [52] propose a Comprehensive Learning Quantum-Inspired Evolutionary Algorithm (CLQEA) by introducing the philosophy of comprehensive learning into quantum-inspired evolutionary algorithms. In CLQEA, each individual in a population learns not only from its own best historical solution searched, but also from the best solutions that other individuals found. This idea has been found very helpful to enhance population diversity through applying a group of elite individuals to perform Q-gates to produce offspring. It is shown that CLQEA outperforms several QIEAs reported in the literature based on extensive experiments carried out on knapsack problems with various items.

As described earlier Nowotniak and Kucharski [22] present the parallel implementation of Quantum-Inspired Genetic Algorithm to solve the knapsack problem in a GPU-based massively parallel computing environment (NVidia CUDA™ technology).

Differential evolution (DE) is a population based evolutionary algorithm widely used for solving multidimensional global optimization problems over continuous spaces. Hota & Pat extend the concept of differential operators with adaptive parameter control to the quantum paradigm and proposes the adaptive quantum-inspired differential evolution algorithm (AQDE) [53]. The performance of AQDE is found to be significantly superior as compared to QEA and a discrete version of DE on the standard 0-1 knapsack problem for all the considered test cases.

Honggang et. al. propose a new algorithm called quantum-inspired ant algorithm (QAA) to solve the knapsack problem [54]. QAA takes the advantage of the principles in quantum computing, such as qubit, quantum gate, and quantum superposition of states, to get more probabilistic-based status with small colonies. By updating the pheromone in the ant algorithm and rotating the quantum gate, the algorithm can finally reach the optimal solution. The detailed steps to use QAA have been presented, and by solving series of test cases of classical knapsack problems, the effectiveness and generality of the new algorithm have been validated.

Patvardhan et al. [55] present design of enhanced Quantum Evolutionary Algorithms and their application for the solution of the Difficult Knapsack Problems (DKPs).

Zhang provides a unified framework and a comprehensive survey of recent work in quantum-inspired evolutionary algorithm [56]. It presents the key ideas related to the multitude of quantum-inspired evolutionary algorithms, sketch the differences between them, survey theoretical developments and applications that range from combinatorial optimizations to numerical optimizations, and compare the advantages and limitations of these various methods. A small comparative study is also conducted to evaluate the performances of different types of quantum-inspired evolutionary algorithms and conclusions are drawn about some of the most promising future research developments in this area.

Zhao et. al. propose a novel approach to adjusting the weightings of fuzzy neural networks using a Real-coded Chaotic Quantum-inspired genetic Algorithm (RCQGA) [57]. Fuzzy neural
networks are traditionally trained by using gradient-based methods, which may fall into local minimum during the learning process. To overcome the problems encountered by the conventional learning methods, RCQGA algorithms have been proposed for adoption because of their capabilities of directed random search for global optimization.

The number of clusters has to be known in advance for the conventional k-means clustering algorithm and moreover the clustering result is sensitive to the selection of the initial cluster centroids. This sensitivity may make the algorithm converge to the local optima. Xiao et. al. propose a quantum-inspired genetic algorithm for k-means clustering (KMQGA) [58]. In KMQGA, a Q-bit based representation is employed for exploration and exploitation in discrete 0–1 hyperspace using rotation operation of quantum gate as well as the typical genetic algorithm operations (selection, crossover and mutation) of Q-bits. Different from the typical quantum-inspired genetic algorithms (QGA), the length of a Q-bit in KMQGA is variable during evolution. Without knowing the exact number of clusters beforehand, KMQGA can obtain the optimal number of clusters as well as providing the optimal cluster centroids. Both the simulated datasets and the real datasets are used to validate KMQGA, respectively. The experimental results show that KMQGA is promising and effective.

Several other researchers are working on quantum-inspired evolutionary algorithm (QEA) to find solutions of different hard and practically difficult problems. Xing et. al. investigate least-cost QoS multicast routing problem in IP/DWDM optical networks, and proposes an improved evolutionary algorithm (AEQEA) [59]. Wang et. al. present a clonal real-coded quantum-inspired evolutionary algorithm (CRQEA) with Cauchy mutation for solving the short-term hydrothermal generation scheduling (SHGS) problem which consider valve-point effects and transmission losses and is a complicated problem which has the characteristics of nonlinear and non-convex [60]. A heuristic search algorithm, the Quantum-inspired Competitive Evolutionary Algorithm (QuCEA), based on both quantum and evolutionary computing, is proposed in [61] which has been applied to Multiple-Fault Diagnosis, a typical NP-hard problem for industrial diagnosis. In [62], Yang et. al. propose a quantum-inspired immune clone algorithm (QICA) and wavelet packet (WP) based adaptive multiscale Bandelet (QICAWP-AMB) for image representation. In [63], an effective hybrid quantum-inspired evolutionary algorithm with differential evolution (HQEDE) is proposed and applied to estimate the parameters of the Lorenz system.
5. PROPOSED WORK

The literature review presented in the previous sections throws up many open issues some of which are proposed to be addressed in the present work.

(i) Quantum-inspired Evolutionary Algorithms (QiEAs) have been proposed for the solution of the Knapsack problems. There is considerable scope for engineering these algorithms with the hybridization of ideas from the other heuristics / approximation algorithms that have been proposed in the literature. An attempt will be made in the present work to design and test hybrid implementations for faster solution of larger problems to obtain optimal or near-optimal solutions.

(ii) Attempts will be made to design and implement algorithms that take advantage of the multi and many core architectures available today for engineering the solution of the more computationally involved Knapsack problems. In this context, many core CUDA machines would be explored. Again the emphasis will be on development of efficient algorithms for the solution of larger problems that attempted hitherto.

(iii) Attempts will also be made to engineer algorithms for the solution of some combinatorial optimization problems from Graph domain viz., max cut, graph bisection etc. by heuristics, approximation algorithms and evolutionary algorithms or their hybrids. Attempts are also proposed to be made engineer CUDA based implementations on larger graphs.

(iv) Focus throughout the study will be on Algorithm Engineering methodology and in providing working algorithms and implementations.

REFERENCES


