A Parallel Genetic Algorithm for 0/1 Knapsack Problem

Yu-Cheng Kang
kevinkang870211@gmail.com
Institute of Information Management, NYCU
Hsinchu, ROC

Man-Ting Chao
manting417@gmail.com
Institute of Information Management, NYCU
Hsinchu, ROC

Yu-Chen Cheng
debbie.jeng0124@gmail.com
Institute of Information Management, NYCU
Hsinchu, ROC

1 INTRODUCTION

In the last fifteen to twenty years, metaheuristics have profoundly changed how we solve these problems and have significantly contributed to addressing complex, hard problem settings efficiently. The knapsack problem is a well-known problem in combinatorial optimization. It also appears in many real-world applications, such as capital budgeting, resource allocation, cargo loading. However, the 0/1 knapsack problem is one of the most famous NP-complete problems. That is, it is unlikely to be solved in polynomial time. Due to the computational complexity of the problem, much research has been put more attention on approximation algorithms. The genetic algorithm (GA) are powerful, domain-independent search heuristic techniques which are inspired by the process of natural selection. Although the genetic algorithm is effective in solving many practical problems, it could be time-consuming during their execution in some huge problems, because there are a lot of candidate solutions that must be evaluated. Additionally, the genetic algorithm repeats the four processes of fitness evaluation, selection, crossover, and mutation until it terminates. During these steps, the algorithm goes through all individuals and uses the same genetic operation. Therefore, this proposal is intended to improve the genetic algorithm performance by data-parallelism approach to solve the 0/1 knapsack problem.

2 STATEMENT OF THE PROBLEM

2.1 0/1 Knapsack Problem

In this proposal, we decided to solve the 0/1 knapsack problem which has been studied extensively in the past few decades. As stated above, because of the complexity of this problem, we are going to solve the 0/1 knapsack problem with the genetic algorithm. In the 0/1 knapsack problem, we are given a set $N = 1, \ldots, n$ of items which are indexed from 1 to n. For each item i, it has a non-negative value v_i and a non-negative weight w_i. In addition, we are given that W is the maximum knapsack capacity. That is, the maximum weight we can carry in the bag. The goal of this problem is to maximize the total value of items in the bag while the total weight of these items does not exceed the knapsack capacity.[5] Therefore, we can formally defined the problem as follows:

$$\text{maximize} \sum_{i=1}^{n} v_i x_i \tag{1}$$

$$\text{subject to} \sum_{i=1}^{n} w_i x_i \leq W, x_i \in 0, 1 \tag{2}$$

3 PROPOSED APPROACHES

This method we use is a type of data-parallelism approach – in other words, a kind of SPMD (Single Program Multiple Data) approach. The flowchart of the genetic algorithm is shown on Fig. 1. It focused on splitting the entire data to small pieces and assigning them to multiple threads to execute an independent genetic algorithm. Each thread would generate their genepool and continue with their individual crossovers and mutations, at which stage each thread would have a child as a result. After all threads have generated a child, they would compare their own children with the current best child and go through several generations of GA. We will select the best choice as the final result at the final stage, which is done in the critical section.

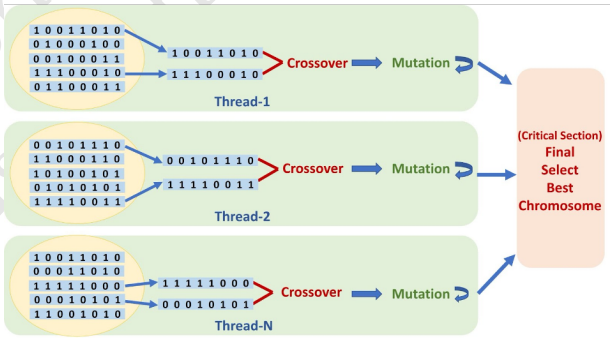

Figure 1: Parallel Genetic Algorithm

4 LANGUAGE SELECTION

We plan to use C++ with OpenMP as our main programming language.

OpenMP, which is based on a fork-join model, has been chosen as a reference parallel programming language for Parallel Genetic Algorithm implementation. OpenMP provides a portable, scalable model for developers of shared memory parallel applications. The API supports C/C++ and Fortran on a wide variety of architectures. It instructs the compiler to organize parallel sections of code in a specific manner and helps to parallelize execution of an application.[6]

The reason to choose OpenMP is that compared with other thread libraries such as pthread, OpenMP is easier to use, it can parallelize serial programs with relatively few annotations that specify where the parallelism region is without many library calls for initialization, synchronization, thread creation, etc.

5 RELATED WORK

5.1 Genetic algorithm

A genetic algorithm is a search heuristic technique inspired by Charles Darwin's theory of natural evolution. In other words, this algorithm reflects the process of natural selection. Generally speaking, genetic algorithms mainly consist of three processes: selection, mutation, and crossover. The flowchart of the genetic algorithm is shown on Fig. 2. To be more specific, a genetic algorithm starts with a set of initial individuals. On every iteration of the algorithm, each individual is evaluated by the fitness function. Then, the fittest individuals are selected to produce new offsprings, that is, generate new search points in a state space of the next generation. The natural evolution of the algorithm continues until one of the termination conditions is satisfied, for example an acceptable solution is found or the computational resources are running out. [2]

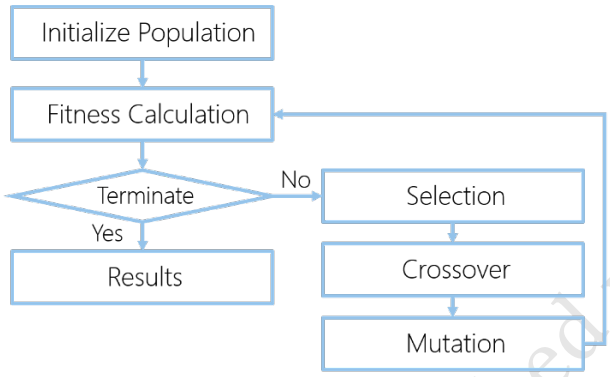

Figure 2: Genetic Algorithm

5.2 Solving the knapsack problem with the genetic algorithm

In this problem, all data is in the public genepool. The idea of chromosome encoding is to have a chromosome consisting of as many genes as there are a total number of items such that each gene index corresponds to item index in the list. Each gene has a value 1 or 0 which means whether the corresponding item is present or not. Once the fitness has been calculated for all population members, we can select fit members to execute crossover and generate their child. In Mutation, which chromosome will undergo mutation is being done randomly. The algorithm terminates if the population has converged.[4] [1] [3]

6 STATEMENT OF EXPECTED RESULTS

With the help of multithread related libraries, we hope we can parallelize genetic algorithm to solve knapsack problem, with approximate optimal solutions, besides and most importantly, reducing the time complexity. Concretely speaking, we will compare 2, 4, 8 or above threads-based cases with the original genetic algorithm, expecting that with the number of threads increases, the executing time the genetic algorithm takes will decrease.

7 TIMETABLE

The following table shows the schedule of our experiment.

Tasks to complete	Approximate time period
Study related papers and find parallel approaches.	2 weeks
Implement GA parallelism code.	1 weeks
Start writing papers and conduct experiments simultaneously.	1 week
Prepare a final report and paper.	1 week

Figure 3: Timetable

REFERENCES

[1] Maya Hristakeva and Dipti Shrestha. 2004. Solving the 0-1 knapsack problem with genetic algorithms. In *Midwest instruction and computing symposium*. 16–17.
[2] Melanie Mitchell. 1998. *An introduction to genetic algorithms*. MIT press.
[3] Rattan Preet Singh. 2011. Solving 0–1 Knapsack problem using Genetic Algorithms. In *2011 IEEE 3rd International Conference on Communication Software and Networks*. 591–595. https://doi.org/10.1109/ICCSN.2011.6013975
[4] Dr. Manish Saraswata Dr. R.C Tripathib. 2020. Solving Knapsack Problem with Genetic Algorithm Approach. *1st INTERNATIONAL CONFERENCE ON INTELLIGENT COMMUNICATION COMPUTATIONAL RESEARCH* (2020).
[5] Wikipedia. 2021. Knapsack problem — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Knapsack%20problem&oldid=1048161179. [Online; accessed 27-October-2021].
[6] Wikipedia. 2021. OpenMP — Wikipedia, The Free Encyclopedia. http://zh.wikipedia.org/w/index.php?title=OpenMP&oldid=64029695. [Online; accessed 27-October-2021].