The Journal of Open Source Software

GKNAP: A Java and C++ package for solving the multidimensional knapsack problem

Shalin Shah[1]

1 Johns Hopkins University

DOI: 10.21105/joss.01756

Software

- Review
- Repository
- Archive

Submitted: 12 September 2019
Published: 18 October 2019

Summary

The 0/1 multidimensional (multi-constraint) knapsack problem is the 0/1 knapsack problem with m constraints. It is a strongly NP-hard problem, and might be difficult to solve using exact methods like branch and bound and dynamic programming, especially when the number of variables is large. We present a genetic algorithm for the multidimensional knapsack problem with Java and C++ code that is able to solve publicly available instances in a very short computational duration. Our algorithm uses iteratively computed Lagrangian multipliers as constraint weights to augment the greedy algorithm for the multidimensional knapsack problem and uses that information in a greedy crossover in a genetic algorithm. The algorithm uses several other hyperparameters which can be set in the code to control convergence. Our algorithm improves upon the algorithm by Chu and Beasley (Chu & Beasley, 1998) in that it converges to optimum or near optimum solutions much faster.

It is possible to use the greedy algorithm as part of a genetic algorithm, and our results show that it works really well. Not only is our algorithm able to exceed the greedy estimate, but for most problem instances, it is able to find the optimum solution. Our algorithm is similar to (Shah, 2019) which uses greedy crossover for the 0/1 knapsack problem. Since the multidimensional knapsack problem has multiple constraints, we assign a weight to each constraint using iteratively computed Lagrangian multipliers. This is similar to the approach in (Chu & Beasley, 1998) which uses surrogate multipliers. The difference is that we use the multipliers in a greedy crossover which is highly constructive and can find optimum solutions much quicker.

Mathematics

We use Lagrangian multipliers to augment the utility ratio for the multidimensional knapsack problem according to the following steps:

For each object and for each constraint (for that object) the weight (constraint) value is multiplied with the corresponding Lagrangian multiplier the sum of these values is obtained. The value obtained is then divided by the number of constraints. Then, the ratio of the value (profit) and the value obtained in the previous step is obtained which is the profit-weight ratio for that object:

$$ratio_i = v_i/((\sum_{j=1}^{m} l_j * w_{ij})/m)$$

where m is the number of constraints, and l_j is the j^{th} Lagrangian multiplier. The greedy crossover simply takes objects from the two parents in non-increasing order of the ratio and constructs one offspring such that it satisfies all constraints.

Our method, as applied to the 0/1 knapsack problem, is similar to the algorithm described in (Shah, 2019). We use techniques like simulated annealing (Kirkpatrick, Gelatt, & Vecchi, 1983) in our work to handle constraints. More details on constraint handling techniques is presented in (Coello, 2002). We generate the initial population with a probability of 0.5. More on this is given in (Hill, 1999).

Traditional evolutionary algorithms are more suitable for problems in which domain specific knowledge is not available. For problems with partial knowledge of the domain, a genetic algorithm, which uses this domain knowledge, is more likely to succeed, as the results clearly indicate. A good search algorithm should be global in nature with a heuristic introduced to give constructive direction to the algorithm. We introduced a new technique of greedy crossover; it forms the core of our genetic algorithm. As the table on the git page shows, our algorithm is able to solve to optimality, all of the instances in a short amount of time. Some problems like Weing7 are harder. Future work could be to run the algorithm on larger instances for which optimum solutions are available. Our algorithm is trivially parallelizable and future work could be to implement the algorithm on Apache Spark or Map-Reduce.

References

Chu, P. C., & Beasley, J. E. (1998). A genetic algorithm for the multidimensional knapsack problem. *Journal of heuristics*, *4*(1), 63–86.

Coello, C. A. C. (2002). Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art. *Computer methods in applied mechanics and engineering*, *191*(11-12), 1245–1287.

Hill, R. R. (1999). A monte carlo study of genetic algorithm initial population generation methods. In *WSC'99. 1999 winter simulation conference proceedings.'Simulation-a bridge to the future'(Cat. No. 99CH37038)* (Vol. 1, pp. 543–547). IEEE.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, *220*(4598), 671–680.

Shah, S. (2019). Genetic algorithm for a class of knapsack problems. *arXiv preprint arXiv:1903.03494*.