Pure and New Mathematics in AI 2024, 1(1), 8052.
https://doi.org/10.24294/pnmai8052

Article

The importance of fine-tuning Gurobi parameters when solving quadratic knapsack problems: A guide for OR practitioners

Dominic Rando[1], Yun Lu[2], Myung Soon Song[2], Francis J. Vasko[2,*]

[1] Computer Science Department, Kutztown University, Kutztown, PA 19530, USA
[2] Department of Mathematics, Kutztown University, Kutztown, PA 19530, USA
*** Corresponding author:** Francis J. Vasko, vasko@kutztown.edu

CITATION

Rando D, Lu Y, Song MS, Vasko FJ. The importance of fine-tuning Gurobi parameters when solving quadratic knapsack problems: A guide for OR practitioners. Pure and New Mathematics in AI. 2024; 1(1): 8052. https://doi.org/10.24294/pnmai8052

ARTICLE INFO

Received: 18 July 2024
Accepted: 12 September 2024
Available online: 11 November 2024

Abstract: In the operations research (OR) literature several highly efficient solution methods for the Quadratic Knapsack Problem (QKP) have been documented. However, these solution approaches are not readily available for industrial applications. In this short paper, we demonstrate that OR practitioners must be careful in their use of general-purpose integer programming software such as Gurobi when solving QKPs. We verify the very positive impact of fine-tuning parameters when solving QKPs with Gurobi.

Keywords: quadratic knapsack problem; general-purpose integer programming software; Gurobi; parameter fine tuning; operations research practitioners

1. Introduction

The Quadratic Knapsack Problem (QKP) has many real-world industrial applications in diverse areas such as telecommunications [1], computer compilers [2], location of freight terminals [3], and wind farm layout optimization [4]. Hence operations research (OR) practitioners need solution methods that will provide guaranteed optimal or near-optimal solutions in a timely and cost-effect manner. Although there are a number of highly efficient algorithms for solving the QKP that appear in the OR literature, their computer codes are not readily available for industrial use. As an example, the highly efficient QKP code QUADKNAP discussed in Caprara et al. [5] comes with the following statement "This code can be used free of charge for research and academic purposes only". Additionally, even if such codes were offered there would be no technical support available. For the OR practitioner to thoroughly understand one of the QKP algorithms documented in the literature, then code it, test and validate it and implement it for industrial use would be very time-consuming and costly. Alternatively, many corporations (as of 12 March 2024 Gurobi claims over 2500 industrial customers spanning 40 industries world-wide) own Gurobi software and routinely solve optimization problems using this software. In this short paper we will demonstrate that when using Gurobi to solve QKPs, parameter fine-tuning is critical to getting proven optimal solutions in a timely manner.

In the next section we will provide a mathematical formulation for the QKP. This will be followed by a brief summary of the best performing algorithms for solving the QKP. Then we analyze how well Gurobi performs when solving QKP instances from the literature—first without and then with parameter fine-tuning. Several observations will conclude the paper.

2. Literature review of algorithms for solving the quadratic knapsack problem

The QKP was first introduced by Gallo et al. [6]. Pisinger [7] provided a comprehensive survey of the QKP. More recently, Cacchiani et al. [8] gave an overview on recent advances dealing with the QKP. After giving the mathematical formulation that appeared in Cacchiani et al. [8], we will focus on the main solution algorithms that have appeared in the literature.

In the QKP we are given a knapsack with capacity c and n items having profit p_j and weight w_j ($j = 1, \ldots, n$). An extra non-negative profit p_{ij} is earned if both items i and j are selected ($i, j = 1, \ldots, n$; $i > j$). The objective function maximizes overall profit, calculated as the sum of the profits of the selected items and of their pairwise profits. This objective is achieved by inserting items into the knapsack such that the capacity is not exceeding. Formally:

$$\max \sum_{j=1}^{n} p_j x_j + \sum_{j=1}^{n-1} \sum_{i=1+j}^{n} p_{ij} x_i x_j$$

$$\text{s.t.} \sum_{j=1}^{n} w_j x_j \leq c$$

$$x_j \in \{0, 1\}\, j = 1, \cdots, n$$

We will briefly review (in chronological order of publication) the main QKP solution algorithms that appear in the literature.

Caprara et al. [5] introduce a branch and bound algorithm (Quadknap) for the QKP, where upper bounds are computed by considering a Lagrangian that is solvable through a series of continuous KPs. Notably, the C code for Quadknap is available for research purposes only. Similarly, Billionnet and Soutif [9] propose a branch and bound algorithm based on a Lagrangian decomposition. Building on these methods, Pisinger et al. [10] develop an exact algorithm that employs a variable fixing procedure, called aggressive reduction, which utilizes the upper bound suggested by Billionnet and Soutif [9], along with another upper bound from Caprara et al. [5], while also integrating several heuristic algorithms to compute lower bounds. Meanwhile, Cunha et al. [11] describe two Lagrangian heuristics based on linear reformulations of the problem, where Lagrangian dual bounds are generated, and their corresponding solutions are applied as input to a primal heuristic. In contrast, Fomeni et al. [12] present a cut-and-branch algorithm in which a sophisticated cutting plane phase is followed by a branch-and-bound phase. Finally, Fomeni et al. [13] introduce a heuristic that combines dynamic programming with a local search procedure, both of which are adapted and implemented in the space of lifted variables of the QKP.

Although the above list is not all inclusive, it serves to demonstrate that there are a significant number of highly specialized solution approaches for the QKP that have been documented in the OR literature. Two more recent papers dealing with the QKP are Fampa et al. [14] and Wu et al. [15]. However, since none of the computer codes associated with these solution approaches are available for industrial use, it would

require significant time and effort to implement one of these solution approaches for an industrial application.

In the next section, using QKP instances available from Beasley's OR Library, we will analyze how well Gurobi can solve these instances; first without any parameter fine-tuning and then with parameter fine-tuning.

3. Empirical results using QKP instances from Billionnet and Soutif [9]

3.1. Description of the QKP instances

It is important to remember that the purpose of this paper is to demonstrate the significant value in using Gurobi's fine-tuning tool specifically when using Gurobi to solve QKPs. Although the are other general-purpose integer programming software packages available, we chose Gurobi because it had performed well on other binary integer programming problems and its tuning tool had performed well for us in the past.

We are not trying to solve all or most of the QKP instances that appear in the literature. This paper is more a proof-of-concept paper instead of a comprehensive computational study. We will consider the impact of parameter fine tuning on larger QKP instances in future work. We downloaded from Beasley's OR-Library the 100 QKP instances that are discussed in Billionnet and Soutif [9]. There are 10 instances for the following combination of number of items and density. Where density refers to the number of non-zero p_{ij} values. For example, if density is 25%, then only 25% of the possible p_{ij} values are non-zero. The number of item-density combinations are: 100-25%, 100-50%, 100-75%, 100-100%, 200-25%, 200-50%, 200-75%, 200-100%, 300-25%, and 300-50% for a total of 100 instances. However, three of the files (100-100%-#4, 200-25%-#3, 300-25%-#3) were corrupted and could not be used. Hence the results are based on 97 QKP instances. Unless otherwise stated, all Gurobi (version 10.0) runs were executed on a PC with the following specifications: Intel® Core™ i3-1005G1 CPU at 1.20 GHz with 8 GB of RAM at 2667 MHz.

3.2. Empirical results using all default parameter values

The authors have successfully solved more than 50,000 binary integer programming problems using general-purpose integer programming software with default parameter values. Among these instances were a large number of hard 0–1 knapsack problems [16] and multidimensional knapsack problems [17]. Hence, it seemed reasonable to try Gurobi with all default parameter values to solve these QKP instances. Using Gurobi in a default mode to solve an integer programming problem means that either a mipgap value less than 0.0001 is achieved or the maximum time has been reached. For a maximization problem mipgap is the difference between the best upper bound minus the best current solution divided by the best current solution [18]. **Table 1** summaries the results of using Gurobi to solve these QKP instances with a maximum execution time of 600s. Results are averaged over problem instances for each number of items and density combination.

Out of the 97 instances only 44 (45%) are solved to proven optimality in 600s or less. The average execution time over all 97 QKP instances was 341.1 seconds. Only the 100 items by 25% density case had all instances solved to proven optimality. When using Gurobi to solve industrial applications it is important that the solutions are either guaranteed to be optimal or that the solutions are guaranteed to be very close to the optimums (very small mipgap values-- usually 0.2% or less). The large final mipgap values in **Table 1** are totally unacceptable to report to management.

Table 1. Results executing Gurobi with all defaults for up to 600s.

Number of variables	Density	Average objective function values	Average time	Maximum time	Average Final mipgap	Maximum final mipgap	Percent Optimal solutions
100	25	32,255	0.5	0.9	0.001%	0.006%	100%
100	50	66,582	246.0	600	2.716%	7.168%	60%
100	75	97,007	361.4	600	9.416%	27.464%	40%
100	100	144,578	337.5	600	4.308%	13.683%	44%
200	25	137,816	335.2	600	5.455%	26.523%	44%
200	50	287,394	481.1	600	9.646%	31.718%	20%
200	75	281,754	422.1	600	10.732%	33.339%	30%
200	100	468,919	492.8	600	26.966%	127.527%	20%
300	25	241,045	342.7	600	16.258%	119.683%	44%
300	50	583,268	396.0	600	5.406%	26.517%	50%
Overall averages		235,905	341.1		9.137%		45%

A simple second step to try to improve the Gurobi results is to increase the maximum execution time. In **Table 2** we show the results of executing Gurobi for a maximum of 1200 seconds. Unfortunately, doubling the execution time had very little impact on the solutions. Notice that there were no changes in any of the 97 objective function values. Only three more proven optimal solutions were obtained. However, the average execution time increased from 341.1 to 664.8 seconds and the average final mipgap was only reduced from 9.137% to 8.831%. Hence, even with an average execution time of over 11 minutes, less than 50% of the instances had obtained proven optimums.

A number of recent papers that have appeared in the literature have reported successfully obtaining tightly bounded solutions quickly on standard PCs. This is achieved by systematically loosening the acceptable terminal mipgap value. This approach is called the simple sequential increasing tolerance (SSIT) methodology and an overview of SSIT is provided in Vasko et al. [19].

Table 2. Results executing Gurobi with all defaults for up to 1200s.

Number of variables	Density	Average objective function values	Average time	Maximum time	Average Final mipgap	Maximum final mipgap	Percent Optimal solutions
100	25	32,255	0.5	1.0	0.001%	0.006%	100%
100	50	66,582	486.3	1200	2.426%	10.553%	60%
100	75	97,007	704.2	1200	8.698%	38.802%	50%
100	100	144,578	671.4	1200	3.922%	12.770%	44%

Table 2. (*Continued*).

Number of variables	Density	Average objective function values	Average time	Maximum time	Average Final mipgap	Maximum final mipgap	Percent Optimal solutions
200	25	137,816	668.8	1200	5.325%	16.255%	44%
200	50	287,394	952.4	1200	9.406%	31.084%	20%
200	75	281,754	842.1	1200	10.400%	32.637%	30%
200	100	468,919	972.2	1200	26.385%	89.499%	20%
300	25	241,045	676.4	1200	16.247%	119.683%	44%
300	50	583,268	677.4	1200	5.060%	26.154%	60%
Overall averages		235,905	664.8		8.831%		48%

Regrettably, because for many of these QKP instances the mipgap values are still very large, the SSIT approach is inappropriate for solving these instances because it cannot be expected to yield tight bounds in short execution times. In the next section we will explore the possible benefit of using Gurobi's parameters fine-tuning tool as a means of obtaining guaranteed near-optimal solutions.

3.3. Background on Gurobi's parameters tuning tool

Gurobi has a list of parameters that all have default values. When you optimize a problem, Gurobi will automatically use the default settings for all parameters. Many parameters have a default setting of −1, meaning that Gurobi will automatically decide how to use that parameter. As an example, take the Cuts parameter. This parameter can have an integer value of {−1, 0, 1, 2, 3} that decides how aggressive the cut generation is. The default setting is −1, and Gurobi describes it as, "The default −1 value chooses automatically." This means that Gurobi will decide how aggressive the cuts should be depending on the problem it is optimizing.

While these parameters provide a tremendous amount of user control, the immense number of possible choices can present a significant challenge when one is searching for parameter settings that improve performance on a particular model. Specifically, since there are 57 parameters that impact the performance of the Gurobi (proprietary) search algorithms, it can be shown that there are more than 2×10^{34} possible parameter settings. Since trying to explore all possible settings is impractical, the purpose of the Gurobi tuning tool is to automate this search. This tool is used to help users choose parameter values that might lead to better performance on a problem. Basically, the user selects a problem to be used as input to the Gurobi parameter tuning tool to try to determine the best parameter values for this particular problem only. This tool is typically executed for at least 12 hours to ensure that high quality parameter values are obtained. More details on this tuning tool are available at [20] an general information on Gurobi parameters is available at [21].

3.4. Empirical results using Gurobi's parameters tuning tool

In order to try to obtain better parameter values to solve our 97 QKP instances, we chose three instances that all had large mipgap values when Gurobi was terminated at 1200 seconds. The Gurobi tuning tool was executed for 12 hours for each of these three instances. The tuned parameter sets determined for these three instances were

then used to solve all 97 QKP instances. We will only provide details for the parameter set that gave the best results for the 97 QKP instances. These optimized parameter values are given in **Table 3**.

Table 3. optimized parameter values based on 200 items by 75% density instance #2.

Parameter Name	Optimized Values	Default Values
MIP Focus	2	0
Gomory Passes	5	−1 (none)
Pre-Q Linearize	2	−1
Branch Dir	−1	0
Cuts	3	−1

In **Table 3** we see that only 5 of the many Gurobi parameters needed to be tuned. We will now give a brief description of each of these five parameters based on information from Gurobi. Additional information is available at [21].

MIP Focus: The MIPFocus parameter allows you to modify your high-level solution strategy, depending on your goals. By default, the Gurobi MIP solver strikes a balance between finding new feasible solutions and proving that the current solution is optimal. If you are more interested in finding feasible solutions quickly, you can select MIPFocus = 1. If you believe the solver is having no trouble finding good quality solutions, and wish to focus more attention on proving optimality, select MIPFocus = 2. If the best objective bound is moving very slowly (or not at all), you may want to try MIPFocus = 3 to focus on the bound.

Gomory Passes: A non-negative integer value indicates the maximum number of Gomory cut passes performed. Overrides the Cuts parameter.

Pre-Q Linearize: Controls presolve Q matrix linearization. Binary variables in quadratic expressions provide some freedom to state the same expression in multiple different ways. Options 1 and 2 of this parameter attempt to linearize quadratic constraints or a quadratic objective, replacing quadratic terms with linear terms, using additional variables and linear constraints. This can potentially transform an MIQP or MIQCP model into a MILP. Option 1 focuses on producing a MILP reformulation with a strong LP relaxation, with a goal of limiting the size of the MIP search tree. Option 2 aims for a compact reformulation, with a goal of reducing the cost of each node. Option 0 attempts to leave Q matrices unmodified; it won't add variables or constraints, but it may still perform adjustments on quadratic objective functions to make them positive semi-definite (PSD). The default setting (−1) chooses automatically.

Branch Dir: Determines which child node is explored first in the branch-and-cut search. The default value chooses automatically. A value of −1 will always explore the down branch first, while a value of 1 will always explore the up branch first.

Cuts: This parameter controls how many cutting planes are generated and are added to strengthen the LP relaxation. More cuts (Cuts = 2 or 3) in the problem also means that solving every single node LP is more expensive. Additionally, adding lots of cuts can also lead to numerical difficulties. Restricting the number of cuts (Cuts =

1 or 0) on the other hand can weaken the LP relaxation and may lead to a higher number of nodes until optimality.

The results of executing Gurobi to solve the 97 QKP instances using the tuned parameter values from **Table 3** are given in **Table 4**.

Table 4. Results executing Gurobi after fine-tuning for up to 1200s.

Number of variables	Density	Average objective function values	Average time	Maximum time	Average mipgap	Maximum mipgap	Percent optimums
100	25	32,255	0.7	1.1	0.000%	0.000%	100%
100	50	66,582	1.3	2.5	0.000%	0.000%	100%
100	75	97,007	3.2	8.4	0.001%	0.001%	100%
100	100	144,578	2.0	7.7	0.001%	0.001%	100%
200	25	137,816	10.1	38.5	0.000%	0.000%	100%
200	50	287,394	27.5	97.1	0.001%	0.001%	100%
200	75	281,754	113.6	802.9	0.001%	0.009%	100%
200	100	468,919	171.9	816.0	0.002%	0.009%	100%
300	25	241,045	25.1	44.4	0.003%	0.009%	100%
300	50	583,268	115.0	385.9	0.001%	0.006%	100%
Overall averages		235,905	48.4		0.001%		100%

From **Table 4** we see that by adjusting just five parameters based on the Gurobi tuning tool recommendation, we not only obtain proven optimums for ALL QKP instances, but the average execution time is only 48.4 seconds! It is important to note that the objective function values for all three of the scenarios discussed are exactly the same. To the OR practitioner, the significant benefit of using the Gurobi tuning tool when solving these QKPs is that guaranteed optimums were obtained for all of these instances. It is important to have high confidence in the model outcomes when major decisions are going to be made based on these results.

Since only five parameter adjustments resulted in such a major improvement in obtaining optimums and reducing execution time, we were curious of the impact of adjusting each of these five parameters one at a time. Because we did not want to incur excessive execution times, we executed the seven scenarios given in **Table 5** for a maximum of 120 seconds per QKP instance. Additionally, we were interested in the impact of setting Pre-Q Linearize equal to 1.

Table 5. Sensitivity results with maximum execution time of 120s.

Parameter adjusted and value	Average objective function value	Average execution time (seconds)	Average final mipgap	Number of proven optimums
All defaults	235,905	72	9.9%	42
MIP Focus = 2	235,902	68	7.3%	49
Gomory Passes =5	235,905	51	5.0%	62
Branch Dir = −1	235,905	76	10.4%	39
Cuts = 3	227,986	64	6.1%	52
PreQLinearize = 2	235,904	21	0.2%	86
PreQLinearize = 1	233,338	42	2.1%	71

From **Table 5** we see that the parameter with the single largest impact on solution quality is when PreQLinearize is set to a value of 2. This is not surprising since we are dealing with quadratic programming problems.

Another takeaway from the table is the difference between option 1 and 2 of PreQLinearize. Gurobi's fine-tuning tool chose the value of PreQLinearize to be set to 2; however, the parameter still has a great effect on the mipgap and number of proven optimums even when set to 1. Interestingly, the average objective function value is much lower than most other runs, and this is due to how each parameter option works. When using option 1, the tree has less nodes to traverse, so each solution found has a great impact on the mipgap. However, since nodes take longer to traverse, Gurobi is not able to find better solutions as quickly. This results in a low mipgap and a low average objective function. On the other hand, Option 2 focuses on cutting the traversal time between nodes, but has no impact on the size of the tree. This modification greatly decreases the mipgap and solve time, suggesting that the solving difficulty stems from the traversal time for each node in the MIP search tree. Linearizing the QKP can have significant benefits. However, when solving a QKP based on these empirical results, we recommend setting the PreQLinerize parameter value to 2.

4. Summary and observations

In this short paper using quadratic knapsack problem (QKP) instances from the literature, we use the general-purpose integer programming software Gurobi to try to obtain proven optimums for these problems. Gurobi default parameter settings fail to obtain optimums for even a majority of these instances in reasonable execution times. However, after using parameter settings recommended by Gurobi's parameter tuning tool, proven optimums were obtained for all QKP instances in an average of only 48 seconds on a standard PC. The single most significant parameter setting for solving these QKPs was setting PreQLinearize = 2.

Based on our experience solving over 50,000 binary integer programming problems, the authors suggest first trying to solve the problems with all default Gurobi parameter values. If these values do not provide desired results, then consider using the Gurobi tuning tool.

Author contributions: Conceptualization, YL, DR, MSS and FJV; methodology, YL, DR, MSS and FJV; software, DR; validation, YL, DR, MSS and FJV; formal analysis, YL, DR, MSS and FJV; investigation, YL, DR, MSS and FJV; resources, DR; data curation, DR; writing—original draft preparation, FJV; writing—review and editing, YL, DR, MSS and FJV; visualization, YL, DR, MSS and FJV; supervision, YL and FJV; project administration, YL. All authors have read and agreed to the published version of the manuscript.

Conflict of interest: The authors declare no conflict of interest.

References

1. Witzgall, C.: Mathematical methods of site selection for electronic message system (ems). Technical Report, NBS Internal Report, 1975.

2. Johnson, E., Mehrotra, A., Nemhauser, G.: Min-cut clustering. 1993. Math. Program. 1993, 62, 133–151.
3. Rhys, J.: A selection problem of shared fixed costs and network flows. 1970, Manag. Sci. 17, 200–207.
4. Quan, N., Harrison, K. M., A tight upper bound for quadratic knapsack problems in grid-based wind farm layout optimization. Eng. Optim. 2024, 50,3, 367-381.
5. Caprara, A., Pisinger, D., Toth, P. Exact solution of the quadratic knapsack problem. INFORMS J. Comput. 1998, 11: 125–137.
6. Gallo, G., Hammer, P.L., Simeone, B. Quadratic knapsack problems, Math. Program. Stud. 1980, 12: 132–149.
7. Pisinger, D. The quadratic knapsack problem — a survey. Discrete Appl. Math. 2007, 155: 623–648.
8. Cacchiani, V., Lori, M., Locvatelli, A., Martello, S. Knapsack problems-an overview of recent advances. Part II: multiple, multidimensional, and quadratic knapsack problems. Computers & Operations Research. 2022, 143. 105693.
9. Billionnet, A., Soutif, E. An exact method based on Lagrangian decomposition for the 0–1 quadratic knapsack problem. European J. Oper. Res. 2004, 157: 565–575.
10. Pisinger, D., Rasmussen, A.B., Sandvik. R. Solution of large quadratic knapsack problems through aggressive reduction. INFORMS J. Comput. 2007, 19: 280–290.
11. Cunha, J.O., Simonetti, L., Lucena, A. Lagrangian heuristics for the Quadratic Knapsack Problem. Comput Optim Appl. 2016, 63: 97-120. https://doi.org/10.1007/s10589-015-9763-3.
12. Fomeni, F.D., Kaparis, K., Letchford, A.N. A cut-and-branch algorithm for the Quadratic Knapsack Problem. Discrete Optimization. 2022, 44: 1-18. https://doi.org/10.1016/j.dispot.2020.100579.
13. Fomeni. F.D. A lifted-space dynamic programming algorithm for the Quadratic Knapsack Problem. Discrete Applied Mathematics. 2023, 335: 52-68. https://doi.org/10.1016/j.dam.2023./02.003.
14. Fampa, M., Lubke, D., Wang, F., Wolkowicz, H., 2020. Parametric convex quadratic relaxation of the quadratic knapsack problem. Eur. J. Oper. Res. 281, 36–49.
15. Wu, Z., Jiang, B., Karimi, H.R., 2020. A logarithmic descent direction algorithm for the quadratic knapsack problem. Appl. Math. Comput. 369, 124854. Xie, X.F., Liu, J., 2007.
16. Song, M. S., Lin, P. H., Lu, Y., Shively-ertas, E., Vasko, F. J. 2024 A practical approach for dealing with hard knapsack problems using general-purpose integer programming software, Intl. Trans. In Op. Res. 1-22. https://doi.org/10.1111/itor.13449
17. Lu, Y., McNally, B., Shively-Ertas, E., Vasko, F. J. 2021."A Simple and Efficient Technique to Generate Bounded Solutions for the Multidimensional Knapsack Problem: a Guide for OR Practitioners", International Journal of Circuits, Systems and Signal Processing. https://doi.org/10.46300/9106.2021.15.178
18. https://support.gurobi.com/hc/en-us/articles/8265539575953-What-is-the-MIPGap.
19. Vasko, F.J., Lu, Y., Song, M.S. Solving hard combinatorial optimization problems with general purpose integer programming software: a guide for OR practitioners. Inside OR. 2023, 627: 13.
20. https://www.gurobi.com/documentation/current/refman/parameter_tuning_tool.html
21. https://www.gurobi.com/documentation/current/refman/parameters.html