Rollout for Dynamic Programming Applied to Integer Nonlinear Programs

Joshua Clugston

Spring 2025

Contents

Introduction

Mathematical models which include integer variables are ubiquitous in many contexts of significant importance within scientific inquiry. Such problems typically are empirically seen as much more difficult to solve than similar models containing only continuous, real-valued variables. The primary difficulty of models with integer variables is that their solution spaces explodes combinatorially as the number of decisions to be made increases. Despite mathematical programs with integer variables suffering from the issue of combinatorial explosion resulting from the presence of integer variables, in the case of problems with linear objective functions and constraints, or problems referred to as *linear programs* (LPs) and *mixed-integer linear programs* (MILPs), several approaches have been developed for obtaining high-quality solutions very quickly with much success. Among the more successful methods specifically for solving MILPs, the most popular of which regularly consider some methodology based on the idea of reducing the feasible region in an attempt to obtain its convex hull, or at the very least an approximation thereof. More generally, however, are integer programs which contain non-nonlinearities either within their objective function or constraints, called *mixed-integer nonlinear programs* (MINLPs). MINLPs are, in general, much more difficult to solve than MILPs. Because of their inherent difficulty, obtaining exact solutions to MINLPs is especially challenging, and typical approaches for obtaining global solutions generally rely on relaxing integrality restrictions and applying a branch-and-bound-like approach to both continuous and integer variables. In contrast to traditional branch-and-bound, additional difficulty with this approach is a consequence of each relaxed problem being a constrained nonlinear program (NLP), which may be non-convex. A common approach for overcoming further difficulty imposed by non-convexity is through the application of convex envelope relaxations, whose goal is to further acquire tight approximations for the original NLP. Branch-and-bound-like methods such as those previously described for MINLPs are regularly restrictive, and empirical performance is often observed to be less than practical for many problems of relevance.

In an attempt to overcome less than ideal performance in terms of computational speed of exact branch-and-bound-like algorithms, heuristics are regularly enlisted as a means of producing approximate solutions very quickly. Some examples of heuristic algorithms for solving MINLPs are those proposed by Fomeni & Letchford (2014) and Fennich et al. (2024), which both utilize a dynamic programming approach in the context of a binary knapsack problem with nonlinearities. Further examples of more general heuristics, particularly designed for solving MINLPs, include sub-NLP and NLP-diving. Both of these approaches attempt to explore the solution space by exploiting the decrease in difficulty obtained from excluding integral restrictions through either fixing integer variables to a constant numerical value, or entirely removing integerality from the model before solving. Between both of the two aforementioned heuristics, the idea is to iteratively perform the relaxation of integerality restriction or fixing of integer variables until a good quality solution is attained. Effectively, each of two latter-most heuristics result in solving NLP subproblems to achieve approximations to the original problem.

One additional possible method for solving potentially large-scale MINLPs is to apply approximations within the traditional dynamic programming scheme, thereby taking an approximate dynamic programming (ADP) approach. To the best of the author's knowledge, however, this approach is not common for MINLPs, despite several methods utilizing ADP for MILPs. For instance, Secomandi (2000) apply neuro-dynamic programming to the vehicle routing problem with stochastic demands, while Bertazzi (2012) use rollout to solve the 0-1 knapsack problem. On the other hand, there is at least one instance of rollout being applied to MINLPs, as done in Bai et al. (2023) for electrical vehicle charging planning. In this document, the goal is to further explore the capability of ADP approaches for solving complicated MINLPs.

Background

Consider the 0-1 quadratic knapsack problem:

$$\text{maximize} \quad \sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij} x_i x_j, \tag{1a}$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_i \leqslant C, \tag{1b}$$

$$\mathbf{x} \in \{0,1\}^n =: \mathbb{B}^n. \tag{1c}$$

Specifically, in (1a)-(1c) above, C is the available space in the knapsack, w_i for $i = 1, \ldots, n$ are non-negative weights associated with inluding item i in the knapsack, and p_{ij} for $i = 1, \ldots, n$, $j = 1, \ldots, n$ are non-negative profits associated with including item i and item j together in the knapsack. By solving (1a)-(1c), the ultimate goal is to maximize profit corresponding to packing items in the knapsack, while simultaneously considering item interaction and ensuring that capacity restrictions are satisfied.

The model described in (1a)-(1c) lends itself to many applications, resulting from its general form. Most popularly, the QKP is well-known for its application to a decision problem utilized for determining optimal satellite station locations. Given its applicability in this context, it should likely not come as a suprise to the reader that the QKP has similarly seen use in the context of other optimal location, planning problems. For instance, QKP has been used in determining the optimal placement of airports and railway and freight handling stations. In addition, some other popular applications of QKP involve compiler design, very large scale integration, and financial applications which, for example, compare optimal investment strategies. It it further observed that QKP is a generalization of a difficult combinatorial problem called the maximum clique problem, which itself has many applications ranging from social network analysis to chemistry.

Despite its general applicability, model (1a)-(1c) has several difficulties which present it as a very challenging problem to solve, especially for instances of practical size. Due to the presence of bilinear terms, $x_i x_j$ for $i = 1, \ldots, n$ and $j = 1, \ldots, n$, which are non-convex, (1a)-(1c) is referred to as a *non-convex mixed-integer nonlinear program.* More precisely, it is observed that the continuous relaxation of (1a)-(1c) is a non-convex NLP. In which case, non-convexity present an especially difficult challenge to resolve due to the potential for local minima to hinder the search for global solutions. Moreover, because (1a)-(1c) can be seen as a generalization of the maximum clique problem, which is known to be NP-hard in the strong sense, it follows that QKP is NP-hard in the strong sense. Therefore, in contrast to the standard 0-1 knapsack problem, unless P = NP, it is unlikely that there exists a pseudo-polynomial time algorithm which may be used to solve QKP. In practice, particular difficulty stems from the data p_{ij}, and solving QKP may be too computationally intensive for moderately sized problems in some cases, provided that these data are restrictive. This latter point will be discussed in more detail in **Section 3.1**.

To overcome some of the difficult, it is common to reformulate QKP to a simpler form. A popular technique regularly employed in global optimization to deal with bilinear terms such as $x_i x_j$ utilizes tight-fitting convex/concave under and over-approximators, called *McCormick envelopes.* Through the introduction of an auxiliary variable $u_{ij} = x_i x_j$ for all $i = 1, \ldots, n$ and $j = 1, \ldots, n$, model (1a)-(1c) may be rewritten in relaxed form as follows:

$$\text{maximize} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} p_{ij} u_{ij}, \tag{2a}$$

$$\text{subject to} \quad u_{ij} \geqslant x_j + x_i - 1, \text{ for all } i \in [n], j \in [n], \tag{2b}$$

$$u_{ij} \leqslant x_i, \text{ for all } i \in [n], j \in [n], \tag{2c}$$

$$u_{ij} \leqslant x_j, \text{ for all } i \in [n], j \in [n], \tag{2d}$$

$$\sum_{i=1}^{n} w_i x_i \leqslant C, \tag{2e}$$

$$\mathbf{x} \in \mathbb{B}^n, \tag{2f}$$

$$\mathbf{u} \in \mathbb{R}_{\geqslant 0}^{n \times n}. \tag{2g}$$

However, because $x_i \in \mathbb{B}$ for all i, the aforementioned relaxed problem is actually an exact reformulation of (1a)-(1c), so solving (2a)-(2g) equates to solving the original QKP formulation. This is especially beneficial since now the continuous relaxation of (2a)-(2g) can be applied to obtain an over-estimate of the solution to the original problem.

An approach for solving (1a)-(1c) first considers instead a relaxation-and-penalization related problem, whereby a non-negative penalty term is added to the original objective function following the relaxation of a complicating constraint. Particularly, if one relaxes the capacity constraint and penalizes its violation using a penalty term $\lambda \in \mathbb{R}_{\geqslant 0}$, the following, so called, *Lagrangian relaxation*

$$\mathcal{L}(\mathbf{x}, \lambda) = \sum_{i=1}^{n} \sum_{j=1}^{n} p_{ij} x_i x_j + \lambda \left(C - \sum_{i=1}^{n} w_i x_i \right) \tag{3}$$

is obtained, in which λ is referred to as a *Lagrange multiplier*. To obtain an approximation for the original problem, it is then necessary to consider solving

$$\text{maximize } \mathcal{L}(\mathbf{x}, \lambda), \text{ subject to } \mathbf{x} \in \mathbb{B}^n. \tag{4}$$

Taking the minimum over $\lambda \in \mathbb{R}_{\geqslant 0}$ in the preceding expression (4) will ultimately yield the solution to QKP, though minimization of (4) is with respect to a non-smooth objective function due the presence of integer variables, so standard methods such as gradient ascent or Newton's method cannot be applied outright in this context. Instead, a basic method which may be used for maximizing is the so called *subgradient method*, whereby subgradients

$$\psi(\mathbf{x}) = C - \sum_{i=1}^{n} w_i x_i \tag{5}$$

are introduced, and an approach similar to gradient ascent is taken by iteratively considering multiplier updates

$$\lambda^{k+1} = \lambda^k + s^k \psi(\mathbf{x}^k) \tag{6}$$

for all $k \in \mathbb{Z}_{\geqslant 0}$. Within each multiplier update, a previous solution $\mathbf{x}$ is obtained by solving (4), and subgradients are scaled according to a positive steps size s^k, which can be defined in multiple ways. Particularly, in what follows, $s^k = s^0$ for some $s^0 \in \mathbb{R}$ for all k. The benefit of considering this approach is that, while solving (2a)-(2g) directly using common methods based on branch-and-bound in the worst case are exponential, solving instead with the subgradient method may be less computationally demanding within this context, especially for difficult problems of reasonably large size.

Method

By considering $r \in [C]$, $k \in [n]$, the original problem is restricted and written as:

$$\max \left\{ \sum_{i=1}^{k} \sum_{j=1}^{k} p_{ij} x_i x_j : \sum_{i=1}^{k} w_i x_i = r,\ \mathbf{x} \in \mathbb{B}^k \right\}, \tag{7}$$

where k denotes the current item's index and r denotes the current amount of remaining space in the knapsack when considering the kth item. Then, similar to before, a subgradient approach is considered, with the restricted problem (7) having restricted Lagrangian relaxation subproblems

$$\max \left\{ \sum_{i=1}^{k} \sum_{j=1}^{k} p_{ij} x_i x_j + \lambda \left(r - \sum_{i=1}^{k} w_i x_i \right) : \mathbf{x} \in \mathbb{B}^k,\ \lambda \in \mathbb{R} \right\}. \tag{8}$$

Let the triple $\mathbf{y} = (k, r, S)$ denote the current state at the kth item with capacity used, r, and set of indices of currently included knapsack items S. The immediate profit earned by considering $\mathbf{y}$ and λ is defined as in Fomeni & Letchford (2014) using

$$g(\mathbf{y}, u) = p_{kk} + 2 \sum_{j \in S \setminus \{k\}} p_{kj}, \tag{8}$$

which is the profit of including item k with capacity used r. A viable rollout approach introduces approximate costs-to-go function

$$\tilde{J}(\mathbf{y}) = \max \left\{ \sum_{i=1}^{k} \sum_{j=1}^{k} p_{ij} x_i x_j + \lambda \left(r - \sum_{i=1}^{k} w_i x_i \right) : \mathbf{x} \in \mathbb{B}^k,\ \lambda \in \mathbb{R} \right\} \tag{9}$$

which is an upper bound on (7), so that approximate $Q-$factors

$$\tilde{Q}(\mathbf{y}, u) = g(\mathbf{y}, u) + \tilde{J}(\mathbf{y}), \tag{10}$$

with $u \in \mathcal{U}(\mathbf{y}) =: \{0, 1\}$ such that $u = 1$ if item k is included in the knapsack while $w_k \leqslant r$, and $u = 0$ otherwise. Using the aforementioned ingredients, a proposed rollout algorithm is therefore described in the following pseudocode.

Rollout for QKP

```
0:  Initialize $\lambda^0$, $s^0$, $S \leftarrow [n]$, $r \leftarrow C$
1:  for $k \in [n]$ do
2:      $\hat{S} = \{i \in S : w_i \leqslant r\}$
3:      $\tilde{i} = \arg\max_{u \in \mathcal{U}(\hat{\mathbf{y}})} \tilde{Q}(\hat{\mathbf{y}})$
4:      $\mathbf{x}^k = \arg\max_{\mathbf{x}} \tilde{J}(\hat{\mathbf{y}})$
5:      $g(\mathbf{x}^k) = r - \sum_{i=1}^{k} w_i x_i$
6:      $\lambda^{k+1} = \lambda^k + s^0 g(\mathbf{x}^k)$
7:      $S = S \backslash \{\tilde{i}\}$, $r = r - w_{\tilde{i}}$
8:      if $r = 0$ or $S = \varnothing$
9:          return $\sum_{i \in S \backslash [n]} \sum_{j \in S \backslash [n]} p_{ij}$
10:     end
11: end
```

Some remarks with respect to the proposed rollout algorithm are in order. First of all, it is important to note that in the **Rollout for QKP** algorithm shown in the preceding, $\hat{\mathbf{y}} = (k, r, \hat{S})$, with $\hat{S} \subseteq S$ denoting the set of feasible item indices for the current state. Second, lines three and four are performed synchronously, for which computation of $\tilde{i}$ is performed and the value $\mathbf{x}^k$ is simultaneously attained. Third, it is clear that the algorithm performance is largely dependent on the difficulty of the embedded optimization problem conducted on line four. Therefore, it is important that a function which is efficiently computed be defined as the approximate cost-to-go function to ensure adequate performance with respect to computational speed. Finally, given the wide applicability of subgradient methods applied to non-convex functionals and rollout to combinatorial problems, the **Rollout for QKP** algorithm is fairly general, and slight modification to the algorithm may still be seen as fruitful for approximating the solutions of other MINLPs. With respect to this latter point, however, it is noted that generally a limitation of the subgradient method is that it can under perform when compared to other techniques for optimizing non-smooth functionals. Future work may instead introduce alternative means for updating either the multipliers or $\mathbf{x}^k$ to further improve performance of the proposed algorithm. Alternative techniques pertaining to the former may explore using bundle methods in place of traditional subgradient methods, for instance.

3.1 Data Generation

Data is generated as normally done within the context of QKP, and follows closely the description provided in Fomeni & Letchford (2014). Precisely, the profits,

$$p_{ij} \sim \begin{cases} 0 & \text{with probability } (1 - \Delta), \\ \mathcal{U}\{1, \ldots, 100\} & \text{with probability } \Delta \end{cases} \tag{11}$$

for each $i = 1 \ldots, n$ and $j = 1, \ldots, n$, where Δ is referred to as the *density*. The density, Δ, is particularly important for determining the difficulty of the problem instance in consideration.

For larger values of Δ, the impact of its effects is observed through the objective function, in which a larger Δ may yield an objective function with greatly many more terms. Consequently, this entails that there more likely will be many interactions that will need to be considered when optimizing, therein resulting in a significantly more challenging problem. On the other hand, the remaining data $w_i \sim \mathcal{U}\{1, \dots, n\}$ for each i, whereas $C \sim \mathcal{U}\{n, \dots, \sum_{i=1}^{n} w_i\}$.

Numerical Results

Comparison was made between the proposed rollout algorithm with base policy as defined in **Section 3**, and by using CPLEX version 1.0.3 to solve the original problem directly. Additionally, a further comparison between the proposed rollout algorithm and solving (2a)-(2g) directly using CPLEX was made. For each comparison, numerical experiments were performed using Julia on a 2020 M1 Macbook Pro with 16 GB of RAM. Data were selected according to **Section 3.1**, with step size $s^0 = 1$ throughout, total number of items and density being considered in the CPLEX comparison for (1a)-(1c) being $n \in \{50, 100, 150\}$ and $\Delta \in \{0.8, 0.65, 0.5\}$, respectively, and $n \in \{50, 100, 200\}$ and $\Delta \in \{0.8, 0.65, 0.8\}$ for the case when CPLEX was applied to solve (2a)-(2g). Default parameters for CPLEX are used for all numerical experiments, with the exception of the time limit in the comparison against solving (2a)-(2g) using CPLEX. In this comparison, a time limit of 600 seconds was imposed.

To compare performance, a "gap" is reported for the proposed rollout method which depends on the solution obtained by CPLEX in each case. Instead of using a gap to determine performance of the rollout method in the traditional sense, the idea is to compare the solution obtained by the proposed rollout method with the solution obtained by the exact method used by CPLEX. Particularly, the gap reported is computed according to

$$\text{“}\texttt{Gap}\text{”} = \frac{|z^r - z^c|}{z^c}, \tag{11}$$

where the objective value for rollout, z^r, and the objective value for CPLEX, z^c, are acquired at termination throughout. Seeing as a time limit is imposed in the McCormick case, the average gap across 10 runs of solving is reported, as opposed to when CPLEX is used to solve (1a)-(1c) directly. The reason for this inclusion is a consequence of the fact that CPLEX may not terminate within the prescribed time limit, as was observed on several runs during numerical experiments. Regardless, across all comparisons made, numerical values such as the number of seconds needed for termination and gap at termination, are reported as the sample average over 10 runs for each method.

Table 1: QKP comparison between the proposed rollout method and CPLEX for varying number of available items and densities.

		Rollout		**CPLEX**
n	**Δ**	**Seconds**	**Gap**	**Seconds**
50	0.8	2.985	0.089	35.346
100	0.65	64.284	0.058	147.883
150	0.5	57.799	0.071	121.692

Table 2: QKP comparison between the proposed rollout method and McCormick for varying number of available items and densities.

		Rollout		**McCormick**	
n	**Δ**	**Seconds**	**Gap**	**Seconds**	**Gap**
50	0.8	4.139	0.104	1.147	0.01%
100	0.65	16.562	0.077	14.351	0.01%
200	0.8	206.114	0.108	342.467	0.383%

From **Table 1** it is seen that the rollout approach consistently outperforms CPLEX in terms of computational speed, while simultaneously obtaining similar quality solutions for each instance considered. However, when comparing rollout with the same moderately sized instances of QKP to the McCormick solution, it is instead observed that CPLEX outperforms rollout on average. Because the improvement over rollout is seemingly minor for the $n = 50$ and $n = 100$ cases, a comparison was further made for a large and difficult instance to see if McCormick's solution with CPLEX scales. In doing so, it was observed that, on average, rollout instead outperformed McCormick, with the variance of both the final solution time between solutions being much larger for McCormick than rollout. In particular, there were many instances in the $n = 200$ case where McCormick reached the time limit before admitting a global solution, whereas rollout consistently terminated with a comparable solution to McCormick with around 200 seconds on average.

Conclusion

As discussed in the previous section, rollout is comparable to solving (2a)-(2g) with CPLEX for QKP instance of moderate size and density, to large density. When considering large and difficult QKP instances, it is clear that rollout scales much better than solving (2a)-(2g) with CPLEX. Thus, rollout appears to be a viable option for a scalable algorithm which may be used to solve large-scale QKP instances. Given the clear improvement in computational speed, accuracy of the proposed approach could potentially be further improved by employing a multi-step lookahead rollout approach. Furthermore, given that McCormick outperforms rollout in some instances, a different base policy could be attempted in later work to improve the results observed. For instance, a potential base policy which may provide additional improvement would consider solving the LP relaxation of (2a)-(2g). Future work should then compare the performance of the proposed approach against other well-known approaches such as those introduced in Fomeni & Letchford (2014) and Fennich et al. (2024), as well as against the approach in which subgradient methods or the LP relaxation of (2a)-(2g) are used. A more extensive study across varying instances could also be conducted to provide more evidence toward the efficacy of each proposed approach.

References

Bai, T., Li, Y., Johansson, K. H., & Mårtensson, J. (2023). Rollout-based charging strategy for electric trucks with hours-of-service regulations. *IEEE Control Systems Letters*, *7*, 2167-2172. doi: doi: 10.1109/LCSYS.2023.3285137

Bertazzi, L. (2012). Minimum and worst-case performance ratios of rollout algorithms. *Journal of Optimization Theory and Applications*, *152*, 378-393. Retrieved from `https://doi.org/10.1007/s10957-011-9902-7` doi: doi: 10.1007/s10957-011-9902-7

Fennich, M. E., Fomeni, F. D., & Coelho, L. C. (2024). A novel dynamic programming heuristic for the quadratic knapsack problem. *European Journal of Operational Research*, *319*(1), 102-120. Retrieved from `https://www.sciencedirect.com/science/article/pii/S0377221724005149` doi: doi: https://doi.org/10.1016/j.ejor.2024.06.034

Fomeni, F. D., & Letchford, A. N. (2014). A dynamic programming heuristic for the quadratic knapsack problem. *INFORMS Journal on Computing*, *26*, 173-182. Retrieved from `https://pubsonline.informs.org/doi/10.1287/ijoc.2013.0555` doi: doi: https://doi.org/10.1287/ijoc.2013.0555

Secomandi, N. (2000). Comparing neuro-dynamic programming algorithms for the vehicle routing problem with stochastic demands. *Computers and Operations Research*, *27*(11), 1201-1225. Retrieved from `https://www.sciencedirect.com/science/article/pii/S030505489900146X` doi: doi: https://doi.org/10.1016/S0305-0548(99)00146-X