An exact dynamic programming algorithm, lower and upper bounds, applied to the large block sale problem

David Nizard\textsuperscript{1}, Nicolas Dupin\textsuperscript{1} and Dominique Quadri\textsuperscript{1}

Abstract—In this article, we address a class of non convex, integer, non linear mathematical programs using dynamic programming. The mathematical program considered may be used to model the optimal liquidation problem of a single asset portfolio. We show the problem can be solved exactly using Dynamic Programming (DP) in polynomial time. We also introduce dedicated heuristics and provide very thin admissible solutions, for medium sized and large instances (for which exact resolution is not efficient). We also derive an upper bound which however is not tight. Lastly, we benchmark our results against a commercial solver, such as LocalSolver [5].

Preamble: Throughout this document, we will often refer to the extended version of this working paper [13] for detailed references, assumptions, proofs or results justifications.

In this article, we are interested in solving a non convex, integer, non linear mathematical program, in a maximization context, with a linear constraint. Those problems are generally NP-hard [10]. Theoretically, their continuous relaxation is not convex, turning the search for a sufficient condition for global optimality into a very arduous endeavor. In practice, very few commercial solvers can handle them. In this study, we used the commercial solver LocalSolver [5] and the free solver NLopt [12].

As mentioned in the abstract, we are interested in the Large Block Sale problem (LBS), also referred to as the optimal Liquidation Portfolio, in discrete time, with integer variables. Our key question, how to optimally split the block into smaller orders in order to maximize the sale proceeds or equivalently minimize overall price impact, has been investigated in founding articles [6], [1], [2]. The corresponding models introduce a stochastic stock price, which accounts for previous trade sizes. On the contrary, in this paper, as in [7], we assume deterministic asset prices. It may seem quite restrictive, from a financial practitioner’s standpoint, but it is in our experience, a very favorable environment to execute large block trades. Moreover, this assumption insulates the market participants response to the liquidation, which constitutes our main object of study, from the evolution of asset prices. Consequently, our resolution algorithms and optimization techniques are valid regardless of the price vector. See [13] for a detailed literature review and discussion regarding our assumptions and problem setup. More recently [8], [4], [14] provide techniques to solve non convex MINLP, and [9] discusses a convex quadratic reformulation. We solve optimally through dynamic programming small instances and introduce thin heuristics for medium size and larger instances.

The main contributions of our work are summarized as follows: (i) We propose a new long term memory model, with a price impact function in $g(\sum_{j=1}^{N} x_j)$, for non linear bounded functions $g$ - (ii) We use dynamic programming to solve our non convex non linear integer program, exactly for small size instances - (iii) We present a two-step method, which yields very tight lower bounds, if not optimal solutions. We also obtain an upper bound, although not tight. (iv) Numerically, we often perform better than LocalSolver (cf. [5]) and achieve a higher optimal coverage ratio. This article is organized as follows. Section II defines the problem and its notations. Then, section III introduces Bellman equation and solves the problem using dynamic programming. In section IV, we present different methods to obtain heuristics and discuss their complexity. In section V, we study the continuously relaxed problem and compute an upper bound. We present preliminary results in section VI. Finally, section VII concludes and presents future works.

II. PROBLEM STATEMENT AND NOTATIONS

We consider an investor in a financial market, who holds $N$ units of an asset and wishes to liquidate them over a given time horizon, split in $T$ time steps. Let $(x_1, \ldots , x_T)$ be decision variables corresponding to the quantity sold at each time $t$. Hence, the admissible decisions set is defined by $\mathcal{D} = \{(x_1, \ldots , x_T) \in \mathbb{N}^T, \sum_{t=1}^{T} x_t = N\}$. We also introduce a strictly increasing function $g$ from $\mathbb{R}$ to $[0, 1]$, such that $g(0) = 0$ and $\lim_{g} g = 1$. Penalty function $g$ models the market response to the investor selling action, taking into account market memory. Let $f$ be the objective function defined by: $f(x) = \sum_{t=1}^{T} [p_t - c_t g(\sum_{k=1}^{t} x_k)] x_t$, where $p_t > 0$ is the asset best bid price and $c_t < p_t$ represents the penalty range. Execution price of $x_t$ units at time $t$ varies between $p_t - c_t$ and $p_t$. Higher volumes lead to lower execution prices, but the rate of decrease reflect market participants information about our intent. Investor’s problem is:

$$\text{OPT} = \max_{x \in \mathcal{D}} f(x)$$

(1)

Problem (1) is well defined, as $\mathcal{D}$ is finite. In the most general case, it is an non convex, non separable, non linear, integer optimization problem.
We also define its continuous relaxation by extending previous notations. Let $C$ be the compact set (proof in [13]) of admissible solutions and $\overline{\mathcal{g}}$, $\overline{\mathcal{f}}$ be the real extensions of previous functions. Continuously relaxed problem is:

$$ UB_1 = \max_{x \in C} \overline{\mathcal{f}}(x) \quad (2) $$

In the remaining of the paper, we will assume $\overline{\mathcal{g}}$ is continuously differentiable. Therefore, problem (2) is also well defined, but not convex in general. For numerical experiments, we select the concave function $\mathcal{g}(x) = \frac{2}{\pi} \arctan(x)$. More examples and sufficient conditions to apply for penalty functions can be found in [13].

III. EXACT DYNAMIC PROGRAMMING ALGORITHM

In this section, we consider the discrete optimization problem $(P_{t,n})$, which is equivalent to problem (1), for a generic $t$ and $n$. Let $O_{t,n}$ be its optimal value.

$$ \max_{(x_1,\ldots,x_t)} \ f(x) = \sum_{i=1}^{t} \left[ p_i - c_i \left( \sum_{k=1}^{i} x_k \right) \right] x_i $$

$$ \text{s.t.} \ h(x) = \sum_{i=1}^{t} x_i - n = 0 $$

$$ \forall i, x_i \in \mathbb{N} $$

$$ \forall i, 0 \leq x_i \leq n \quad (3) $$

Theorem 1: (Bellman Equation): $\forall t, n, 1 \leq t \leq n$, $O_{t,n} = \max_{i \in [0,n]} \left\{ O_{t-1,n-i} + \left[ p_t - c_t \mathcal{g}(n) \right] i \right\}$, with initial conditions $O_{t,0} = O_{0,n} = 0$. Theorem 1 provides an explicit scheme, described below, to solve problem (1), through dynamic programming (cf. [3]).

Algorithm 1: Exact Dynamic Programming

\[
\text{for } t = 1 \text{ to } T \text{ do} \\
\quad \text{for } n = 1 \text{ to } N \text{ do} \\
\quad \quad \text{for } k = 0 \text{ to } n \text{ do} \\
\quad \quad \quad O_{t,n} = \max \left( O_{t-1,n-k} + \left[ p_t - c_t \mathcal{g}(n) \right] k \right) \\
\quad \text{compute optimal strategy by backtracking} \\
\quad n = N \\
\text{for } i = T \text{ to } 1 \text{ do} \\
\quad \text{if } O_{t,n} = O_{t-1,n} \text{ then} \\
\quad \quad x_i = 0 \\
\quad \text{else} \\
\quad \quad k = n \\
\quad \quad \text{while} \left( O_{t,n} \neq O_{t-1,n-k} + \left[ p_t - c_t \mathcal{g}(n) \right] k \right) \wedge (k > 0) \\
\quad \quad \quad k = k - 1 \\
\quad \quad \quad x_i = k \\
\quad \quad \quad n = n - k \\
\]

Complexity: $O(TN)$ in space (memory) and $O(TN^2)$ in time. Time complexity is polynomial in $(T + N)$, cubic at worst. However, in practice, exact resolution proves too costly in space/time for large instances.

IV. LOWER BOUNDS OF THE INITIAL PROBLEM

In this section, we present different methods to rapidly compute tight lower bounds.

A. Naive heuristics

We selected two intuitive heuristics, which we will use as benchmark for calibration and numerical experiments:

1. The fire sale $x = (M, 0, \ldots, 0)$; we liquidate the block in one shot at the first time step.
2. The uniform sale $x = (N/T, \ldots, N/T)$, which liquidates the block evenly with time.

B. Two-step DP based methods

This technique consists in two independent steps:

- **Coarse grain DP**: the first stage consists of selling buckets of $P$ units ($P \gg 1$), which we refer to as grain P. We apply algorithm 1 with $N' = \lfloor N/P \rfloor$. Hence its time complexity is $O\left( \frac{T N^2}{P^2} \right)$, faster by a factor $P^2$. The coarser the grain the faster the heuristic, but the lesser its quality (distance to optimal).

- **DP with bounds**: starting from a good admissible solution, we apply the exact DP algorithm in its neighborhood and restrict the search by imposing bounds on admissible solutions. Let $x^0$ be the initial solution, and $l$ and $u$ the lower (resp. upper) bounds for $x$ such that: $\forall t 0 \leq l_t \leq x_t \leq u_t \leq N$. Let $L_t = \min(\sum_{i=1}^{t} l_i, N)$ and $U_t = \min(\sum_{i=1}^{t} u_i, N)$; Hence, $\forall t L_t \leq y_t \leq U_t$, where $y_t = \sum_{i=1}^{t} x_i$. We then rewrite the restricted Bellman equation and introduce the DP with bounds algorithm 2:

Algorithm 2: Dynamic Programming with bounds

\[
\text{for } t = 1 \text{ to } T \text{ do} \\
\quad \text{for } n = L_t \text{ to } U_t \text{ do} \\
\quad \quad \text{for } k = l_t \text{ to } u_t \text{ do} \\
\quad \quad \quad O_{t,n} = \max \left( O_{t-1,n-k} + \left[ p_t - c_t \mathcal{g}(n) \right] k \right) \\
\quad \quad \text{backtracking is identical to Algorithm (1)} \\
\quad \text{Get solution by backtracking in } O_{T,N} \text{ matrix} \\
\]

Complexity: we assume our research neighborhood $u_t - l_t$ is bounded by some $R \geq 0$ for all $t$. Then, space complexity becomes $O\left( T^2 R \right)$ and time complexity is $O\left( T^2 R^2 \right)$. One may notice these two steps can work in sync. For instance, We firstly compute a $P$ grain solution and then refine it using DP with bounds algorithm in a $\lambda P$ size funnel. They can also be run independently as any good solution can be improved using DP with bounds algorithm. We can even start from a continuous solution of problem 2 and use $l_t = \max(0, \lfloor x_t^0 \rfloor - \lambda P)$, $u_t = \min(\lfloor N \rfloor, \lceil x_t^0 \rceil + \lambda P)$ as bounds. However, the contrary to exact DP, we can not guarantee results are optimal. It is the main drawback of the two-step method.
C. Iterated Local Search (ILS)

In this section, we introduce an ILS, described below as algorithm 3, which starts from an admissible solution and provides an admissible local maximum. Let \( x^0 \) be an admissible solution. We first shift \( x_i^0 \) by \( +P \) and \( x_i^{0+1} \) by \( -P \), for some \( i \) and integer \( P \). We apply these shifts only if resulting decision variables remain in the feasible \([0; N]\) domain. We can apply reverse shifts and make \( i \) vary. We store the best solution. And so on until we reached a fixed point which corresponds to the local maximum returned in output. Regarding its application, it can be used, either as a first step heuristic, starting from a naive solution (e.g. uniform sale), or as a second step to improve an existing local maximum.

D. Commercial discrete solver: LocalSolver

We selected LocalSolver v9.5 (Linux64, build 20201030), as one of the leading solvers to benchmark against our lower bounds.

E. Free continuous solver NLopt

We saw in section IV-B that continuous solutions can be used in our two-step approach. They are computed in this paper, thanks to the free/open-source solver NLopt v2.6.2, which specializes in continuous nonlinear optimization. We selected the Conservative Convex Separable approach [15], in its quadratic version (CCSAQ), which empirically works best for the relaxed problem 2. A more detailed discussion can be found in [13].

V. Upper bound using monotony

In this section we provide an upper bound of problem (1) under some convexity conditions on \( f \). We first notice that penalty function \( g \) is assumed strictly increasing, and decision variables \( x_i \)'s are non negative, hence: \( f(x) \leq U(x) = \sum_{t=1}^{T} \left[ p_t - c_t \cdot g(x_t) \right] x_t \). We also define the problem:

\[
UB_2 = \max_{x \in \mathbb{C}} U(x)
\]

where

\[
U(x) = \sum_{t=1}^{T} u_t(x_t) = \sum_{t=1}^{T} \left[ p_t - c_t \cdot g(x_t) \right] x_t
\]

Problem (4) is well defined, and its value function is separable (it can be written as a sum of univariable functions). We now introduce sufficient condition on \( g \) to ensure problem (4) is concave.

\[
\text{Lemma 1: Let function } x \mapsto x g(x), \text{ defined in } \mathbb{R}_+, \text{ be strictly convex. Then function } U \text{ is strictly concave and problem (4) is concave.}
\]

\[
\text{Lemma 2: Concave functions selected for numerical experiments (cf. section II): } g(x) = \frac{2}{\pi} \arctan(x) \text{ satisfy the assumptions of lemma 1.}
\]

A. Resolution of the separated problem

We compute the Lagrangian function and solve it for stationary points. Constraint qualification condition is satisfied by linearity of the constraint, and problem 4 is convex. Therefore, resolution of Lagrange equations provide a global maximum.

\[
L(x, \lambda) = \sum_{t=1}^{T} \left[ p_t - c_t \cdot g(x_t) \right] x_t - \lambda \left( \sum_{t=1}^{T} x_t - N \right)
\]

\[
\nabla L(x, \lambda) = 0 \iff \left\{ \begin{array}{l}
\forall t, \left[ x g'(x_t) \right] = \frac{p_t - \lambda}{c_t} \\
\sum_{t=1}^{T} x_t = N = 0
\end{array} \right.
\]

We determine sufficient conditions to solve the Lagrangian equations, when the price vector \( p \) and consequently vector \( c \) are constant.

\[
\text{Lemma 3: Under the assumption of Lemma 1, if the price vector } p \text{ and consequently vector } c \text{ are constant, then the optimal strategy is } \dot{x} = \left( \frac{N}{T}, \ldots, \frac{N}{T} \right) \text{ and global maximum is } UB_2 = N \left[ p - c g \left( \frac{N}{T} \right) \right]
\]

When price are not constant, we can still get an upper bound. Indeed, let \( \overline{p} = \max_t p_t \) and \( \underline{c} = \min_t c_t \).
Lemma 4: Under Lemma 1 assumptions, with \( p = \max_t p_t \) and \( c = \min_t c_t \), the following inequality holds:

\[
UB_2 \leq N \left[ p - c \left( \frac{t}{N} \right) \right] = UB_2
\]

We note, \( UB_2 \) and \( UB_2 \) are equal when prices are constant. Hence, we refer to the latter in numerical experiments.

VI. NUMERICAL EXPERIMENTS

In previous sections, we described techniques to obtain lower and upper bounds. We now study their numerical performances in terms of quality and CPU time. This section is organized as follows.

In paragraph VI-A, we detail the numerical experiment design for results reproducibility. We describe the machine characteristics, problem parameters selection and the price vector simulations. We also discuss the penalty function calibration process and its underlying motivations.

Then in paragraph VI-B, we present our results for small and medium size instances. We start by introducing metrics, problem parameters selection and the price vector simulations. We present the exact resolution via DP and discuss its applicability. Then we present aggregated results for small and medium size instances, with both lower and upper bounds. In paragraph VI-C, we move on to large instances for which exact resolution is not available. We shortly discuss time and memory limitations of our algorithms. We present quality and CPU time results providing a gap, although not tight, for the optimal value of the initial problem \( (1) \), when both lower and upper bounds are available. As for small and medium instances, we present representative examples to show the influence of stock price variation.

A. Experiment setup and penalty function calibration

**PC characteristics:** numerical experiments were run using a PC with Intel Xeon(R) Silver 4114 at 2.20 Ghz , 2 sockets, 20 core, and 32 Gb of RAM. O/S is Linux Ubuntu 18.04 (Bionic Beaver) and c++ compiler gcc v9.3.0

**Problem size** \( (T,N) = (10^6,10^9) \), \( a < b \), with \( 1 \leq a \leq 3 \) and \( 2 \leq b \leq 9 \) are labeled in the results tables. In particular, \( T \) divides \( N \), as there is no specific interest to deal with odd blocks. When there is no ambiguity in the results tables, we further simplify notations by ignoring the base and write \( (T,N) = (a,b) \).

**Minimum sale price** \( c_t \): we assume \( c_t = \beta p_t \), with \( 0 < \beta < 1 \) constant. For numerical experiments, we set \( \beta = 0.9 \) It means the minimum sale price for largest volumes (even a fire sale) is equal to 10% of the price without penalty.

**Stock price dynamic:** although deterministic, we simulate stock prices using a classical Geometric Brownian Motion stochastic process as in [11], starting at \( p_0 = 100 \), with moments \( (\mu, \sigma): \frac{dS_t}{S_t} = \mu dt + \sigma dW_t \)

**Penalty function calibration:** \( g \) is bounded. \( g(0) = 0 \) and \( \lim_{t \to \infty} g = 1 \). We aim to link the upper limit to the block size \( N \). When traded size \( x \) is close to \( N \), the penalty ratio \( g(x) \) should get close to 1. In practice, we set a threshold \( H \) such that, \( g(\eta N) = H \) and thus determine \( \eta \). We then use the calibrated function \( x \mapsto g(\eta x) \) as penalty function in numerical experiments. Further details on data calibration can be found in [13].

B. Numerical Experiments for small and medium instances

In this section, we first describe table labels and then present the resolution for small and medium instances.

**Table notations:** CST refers to constant prices and AVG to average over the different simulated price vectors. When results differ materially between price vectors, we mention it explicitly. In results tables measuring bounds quality, figures are expressed in percentage and measure the relative difference to optimal coming from exact DP. For CPU time tables, time is measured in seconds, \( \epsilon = 0.01 \) corresponds to the minimum numerical value and is expressed with its corresponding unit (% or seconds). DNC means the algorithm either did not converge within allowed time of 10 minutes, or returned an out of memory error.

**FS** refers to the naive fire sale heuristic, **US** to the uniform sale, **TS1** stands for two-step with coarse grain DP, **TS2** relates to two-step with NLOpt heuristic, **ILS** corresponds to the discrete gradient described in section (IV-C) and initialized with uniform sale. Finally, **LS** refers to LocalSolver ran with approximately the same time limit as the best lower bound (capped to 10 minutes). **UB** corresponds to the upper bound \( UB_2 \)

**Instances size:** for numerical experiments, we will consider the instance size as:

Small, if \( (T,N) \in \{ (1,2), (1,3), (1,4), (2,3), (2,4) \} \)

Medium, if \( (T,N) \in \{ (1,5), (1,6), (2,5), (3,5) \} \)

Large, if \( (T,N) \in \{ (1,7), (1,8), (1,9), (2,6), (2,7), (2,8) (2,9), (3,6) \} \)

Very large, when \( (T,N) \in \{ (3,7), (3,8), (3,9) \} \)

We now solve exactly problem \( (1) \) for small and medium size instances using exact DP algorithm from section III. Time complexity, displayed in table 1, is \( O(T^2 N^3) \) as expected. When \( (T,N) \geq (2,5) \) (in a general sense), exact resolution takes from a few hours to a few days. Hence, exact DP is not tractable for for those instances.

Then, we discuss two-step method results. As discussed in section IV, we run a DP with grain \( P \) and then refine

**TABLE I**

| CPU | CST | AVG |
|-----|-----|-----|
| \( 10^4 \) | \( 10^4 \) | \( < \epsilon \) |
| \( 10^4 \) | \( 10^4 \) | \( 0.02 \) |
| \( 10^4 \) | \( 10^4 \) | \( 1.78 \) |
| \( 10^4 \) | \( 10^4 \) | \( 179 \) |
| \( 10^4 \) | \( 10^4 \) | \( 0.18 \) |
| \( 10^4 \) | \( 10^4 \) | \( 17.68 \) |
| \( 10^4 \) | \( 10^4 \) | \( 1866 \) |
| \( 10^4 \) | \( 10^4 \) | \( 17576 \) |
| \( 10^4 \) | \( 10^4 \) | \( 18261 \) |

-149-
the solution with the same $\lambda P$ size funnel. We computed results for $P = 100$ and $\lambda = 5$. We achieved a coverage ratio (number of instances where the optimal is reached divided by total number of instances) higher than 98% on about 110 instances. By contrast LocalSolver coverage is about 10%. More details about coverage ratio and two-step complexity can be found in [13].

In tables II and III above, we present aggregated results, for all our algorithms applied to small and medium size instances. With the notable exception of ILS, results profile are similar for CST and AVG. As expected, naive heuristic FS and US yields the worst lower bounds. LocalSolver returns a very good lower bound for every instance, but rather seldom the optimum. TS1 and TS2 reached the optimum almost every time for small and medium size instances. The upper bound UB, on the other hand, is optimum almost every time for small and medium size instances. The upper bound UB returns a very good lower bound for every instance, but rather seldom the optimum. TS1 and TS2 reached the optimum almost every time for small and medium size instances. The upper bound UB, on the other hand, is optimum almost every time for small and medium size instances.

In tables II and III above, we present aggregated results, for all our algorithms applied to small and medium size instances. With the notable exception of ILS, results profile are similar for CST and AVG. As expected, naive heuristic FS and US yields the worst lower bounds. LocalSolver returns a very good lower bound for every instance, but rather seldom the optimum. TS1 and TS2 reached the optimum almost every time for small and medium size instances. The upper bound UB, on the other hand, is optimum almost every time for small and medium size instances. The upper bound UB returns a very good lower bound for every instance, but rather seldom the optimum. TS1 and TS2 reached the optimum almost every time for small and medium size instances. The upper bound UB, on the other hand, is optimum almost every time for small and medium size instances.

C. Numerical Experiments for large instances
At this scale, exact DP is not available. Moreover several algorithms do not converge for (very) large instances in the allowed time or are subject to memory constraint. We start by discussion these limitations.

Memory limitations and potential improvements: a double takes 8 bytes in the heap memory and space complexity of DP based algorithms is in $O(TN)$. For our 32Gb RAM computer, the limit is $\frac{\ln(TN)}{\ln(2)} \approx 9.63 < 10$. Therefore, we are limited to $(T, N) = (1, 8), (2, 7)$ or $(3, 6)$ for two-step methods. DP with bounds algorithm, for a $R$ size funnel, is in $O(T^2 R)$. So a leaner data structure could save memory in the order of $\frac{T N}{2 R^2} = \frac{N}{2 R T}$, which is significant for (very) large instances. We leave it as a perspective in section VII.

Large instances tables presentation: in the above bound quality tables IV and V, the best lower bound for each instance is specified in the third column, against which quality is defined, as the relative value to the best known lower bound.

For large instances, in both constant and average cases TS1 is the best lower bound, whenever available. TS2 is $\epsilon$ close to TS1, but not better. As for medium size instances, naive heuristics are underperforming and the upper bound is not
tight, ranging from 40% to seven fold. Hence the optimal interval $[\text{Best LB}; \text{UB}]$ remains large. Specifically for the constant case, every lower bound is close, within a 0.1% radius. ILS is the best lower bound when two-step approach is not available as it outperforms LocalSolver. However for the average (non constant) case, quality difference is more pronounced and LocalSolver becomes the best lower bound, when two-steps are not available. Similar to previous section VI-C, LS and ILS vary significantly among instances, the largest gaps being observed for high volatility stock prices.

**CPU time analysis:** in the above table VI, TS1 and TS2 run, when available, in tractable time for large instances. TS2 is faster than TS1. ILS is in general fairly fast, except for very large instances, where, in the constant case, where it becomes intractable. Naive heuristics and UB are computed in constant time. More details can be found in [13].

### VII. Conclusion and Perspectives

We solved the non convex, integer, non linear mathematical program (1), with a linear constraint, exactly for small instances using dynamic programming. We also found either the optimal or a very tight lower bound for medium sizes instances thanks to the two-step method based on hybrid DP (coarse grain or continuous relaxation coupled with DP with bounds). We provided different approaches to get tight lower bounds for medium size and some large instances. Numerically, we beat LocalSolver in most cases. We also obtain an upper bound which is not tight.

For some large and very large instances, where two-step method cannot be applied due to memory constraints, leaving the Iterated Local Search our only option. In that case, we beat LocalSolver in the constant case only, and underperform it in the average case. Our upper bound provides us again with a wide interval for the optimal value.

We now discuss the shortcomings of our approaches and the perspectives for future work. We begin with a technical improvement and then present methodological perspectives for future research.

- **Memory management:** the efficient data structure suggested in section VI-C would enable to improve memory management and potentially gain an order of magnitude.

- **Upper bound quality:** our upper bound is not tight and gets wider when $T$ grows. A better upper bound would provide a tighter gap for the optimal value, especially for (very) large instances.

- **Iterated Local Search:** although ILS algorithm returns very good results in the constant case, it does not fare well when prices fluctuations are wild. While shifting adjacent time steps is not efficient in that case, one can shift $x_{i+k}$ for arbitrary $k$. An interesting question is how to choose the sequence of $k$’s to improve quality in reasonable CPU time.

### References

[1] Robert Almgren and Neil Chriss. Optimal execution of portfolio transactions. *Journal of Risk*, 3, 2000.

[2] Robert F. Almgren. Optimal execution with nonlinear impact functions and trading-enhanced risk. *Applied Mathematical Finance*, 10(1):1–18, 2003.

[3] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[4] Pietro Belotti, Christian Kirches, Sven Leyffer, Jeff Linderoth, James Luedtke, and Ashutosh Mahajan. Mixed-integer nonlinear optimization. *Acta Numerica*, 22:1–131, 2013.

[5] Thierry Benoist, Frédéric Gardi, Julien Darlay, and Romain Megel. Localsolver, 2020. https://www.localsolver.com/home.html.

[6] D. Bertsimas and A. Lo. Optimal control of execution costs. *Journal of Financial Markets*, 1, 1998.

[7] Stephen Boyd, Enzo Busseti, Steven Diamond, Ronald N Kahn, Kwangmoo Koh, Peter Nystrup, and Jan Speth. Multi-period trading via convex optimization. arXiv preprint arXiv:1705.00109, 2017.

[8] Samuel Burer and Adam N Letchford. Non-convex mixed-integer nonlinear programming: A survey. *Surveys in Operations Research and Management Science*, 17(2):97–106, 2012.

[9] Sourour Elloumi, Anelie Lambert, and Amaud Lazare. Solving unconstrained 0-1 polynomial programs through quadratic convex reformulation. *Journal of Global Optimization*, 80(2):231–248, 2021.

[10] Michael R Garey and David S Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. WH Freeman & Co., 1979.

[11] John C. Hull. *Options, futures and other derivatives*. Pearson Prentice-Hall, Upper Saddle River, NJ, USA, 5th edition, 2002.

[12] Stephen G Johnson. The nlopt nonlinear-optimization package, 2021. http://github.com/stevengj/nlopt.

[13] David Nizard, Nicolas Dupin, and Dominique Quadri. An exact dynamic programming algorithm, lower and upper bounds, applied to the large block sale problem. arXiv preprint:2112.12893, 2021.

[14] Martin Schmidt and Yasmine Beck. Spring school on "minlps and bilevel problems", lectures notes, June 2021, Lamsade, Université Paris-Dauphine.

[15] Krister Svanberg. A class of globally convergent optimization methods based on conservative convex separable approximations. *SIAM journal on optimization*, 12(2):555–573, 2002.