A multiplicative weights update algorithm for MINLP

Luca Mencarelli[1], Youcef Sahraoui[1,2], Leo Liberti[1]

[1] *CNRS LIX, École Polytechnique, F-91128 Palaiseau, France*
Email:{liberti,mencarelli,sahraoui}@lix.polytechnique.fr

[2] *OSIRIS, EDF R&D, F-92141 Clamart, France*
Email:youcef.sahraoui@edf.fr

May 4, 2016

Abstract

We discuss an application of the well-known Multiplicative Weights Update (MWU) algorithm to non-convex and mixed-integer nonlinear programming. We present applications to: (a) the distance geometry problem, which arises in the positioning of mobile sensors and in protein conformation; (b) a hydro unit commitment problem arising in the energy industry, and (c) a class of Markowitz' portfolio selection problems. The interest of the MWU with respect to one of its closest competitors (classic Multi-Start) is that it provides a relative approximation guarantee on a certain quality measure of the solution.

1 Introduction

The Multiplicative Weights Update (MWU) algorithm [2] is a stochastic heuristic with a relative performance guarantee on a weighted average of the errors. Its typical application is to decide the best way to take advice from a set of advisors which repeatedly express different opinions. At each iteration, the MWU performs the following tasks: (i) it updates a probability distribution on the advisors' performance, based on gains/costs at the preceding iteration; (ii) it samples new decisions from the current distribution; and (iii) it updates the gains/costs vector based on the decisions. The MWU yields an approximation guarantee on the weighted average gains/costs relative to the performance of the best advisor.

The MWU was used in the past to derive approximation algorithms for Linear Programming (LP) [33] and Semidefinite Programming (SDP) [1]. In this paper, we describe the computational application of the MWU to two novel settings: nonconvex Nonlinear Programming (NLP) and Mixed-Integer Nonlinear Programming (MINLP). Although we do not derive an approximation algorithm, we demonstrate the applicability of the MWU as a heuristic for NLP and MINLP. We showcase its application to three well known problems: the Distance Geometry Problem (DGP) [29], a variant of the Hydro Unit Commitment (HUC) problem from the energy industry [40], and a variant of Markowitz' Mean-Variance Portfolio Selection Problem (MVPS) [32]. We show that the MWU still retains its (relative) approximation guarantee in these new settings, whilst generally performing as well as, or better than, a classic Multi-Start (MS) approach (see Alg 1). Note that some work is necessary to adapt MS to any given optimization problem.

Algorithm 1 Multi-Start

1: **while** termination condition is not met **do**
2: sample a starting point x'
3: perform a local descent from x', yielding $\tilde{x}$
4: if $\tilde{x}$ improves the best optimum x^* so far, update x^* with $\tilde{x}$
5: **end while**

We must determine the best distribution to sample from, what kind of local descent algorithm to employ,

and which termination conditions are appropriate. We shall see that the MWU heuristic applies generally to optimization problems in a similar way: a nontrivial amount of work is necessary to adapt it to the given problem.

Our main motivation for singling out the MWU as a good heuristic is that, unlike the vast majority of heuristics, the MWU comes with a relative approximation guarantee on some user-defined cumulative (over all MWU iterations) mean costs (or errors) $\psi_1, \dots, \psi_q$, which turn out to be bounded above by a piecewise linear function of the cumulative (again, all over MWU iterations) *lowest* cost $\min_{i \leq q} \psi_i$. Although, in general, this is a rather weak guarantee, it is surprising that it should exist at all, considering the generality of the method. The MWU readily turns into an approximation algorithm whenever the structure of the problem allows one to provide a guaranteed upper bound to the lowest cost. Although we do not derive such approximations in this paper, we mean to study this issue in further works, and hope that this paper will spark interest in the matter.

1.1 The MWU algorithm

We believe that Arora et al.'s excellent survey [2] provides an introduction to the MWU that we cannot hope to improve as regards clarity: we therefore borrow from it here to introduce the MWU by means of an application example.

An investor employs q advisors to help her decide every day whether the price of a given stock will increase or decrease. We let θ_i be the prediction of the i-th advisor, for each $i \leq q$. The investor maintains a weight vector $\omega \in [0,1]^q$, where the component ω_i is used to weigh the reliability of the i-th advisor, for each $i \leq q$. At the outset, all weights are initialized to 1. As the days pass, each wrong prediction is penalized by a unit cost ψ_i to be born by the i-th advisor, whereas correct predictions have zero cost. Note that we can also assign gains, represented as negative costs; and, more generally, we consider a cost vector $\psi \in [-1,1]^q$, where ψ_i denotes the cost, or gain, to be attributed to advisor i (for $i \leq q$). At the end of the day, the weights are updated as follows:

$$\forall i \leq q \quad \omega_i \leftarrow \omega_i(1 - \eta\psi_i), \tag{1.1}$$

where η is a given positive constant $\leq \frac{1}{2}$. The next day, each prediction θ_i is weighed by a random number sampled in $[0, \omega_i]$ (for $i \leq q$), and the investor makes her decision based on the weighted average of the predictions. The weights ω should be thought of as a discrete distribution over $\mathcal{Q} = \{1, \dots, q\}$ which is updated every day. As such, we define their normalized version as $p = \omega / \|\omega\|_1$.

More formally, given positive constants $\eta \leq \frac{1}{2}$ and $T \in \mathbb{N}$, the MWU maintains a list of weights $\omega = (\omega_i \mid i \leq q) \in [0,1]^q$ which are used to randomly update the values of a vector $\theta \in \Theta \subseteq \mathbb{R}^q$ of decision variables related to an associated vector $\psi \in [-1,1]^q$ of costs in a componentwise fashion; these costs are then used to update the weights ω according to Eq. (1.1) before starting the next iteration. This procedure is repeated for T iterations (see Alg. 2). The idea is that the weights ω iteratively adapt

Algorithm 2 (Didactical) Multiplicative Weights Update

```
while termination condition is not met do
    weigh each prediction θ_i by p_i
    compute the cost vector ψ associated to θ
    update ω using ψ as in Eq. (1.1)
end while
```

θ to being a good solution of an optimization problem which aims at minimizing a weighted average of the costs ψ. As the i-th cost ψ_i gets smaller (and perhaps negative, becoming a gain), the associated weight ω_i increases (because of Eq. (1.1)). One (heuristically) hopes that this will yield an even smaller cost at the next iteration. As the MWU proceeds for T iterations, we index ω, p and ψ by $t \leq T$. Thus,

the expected cost of the process on day $t \leq T$ is $p^t \psi^t$, and the cumulative expected cost over the time horizon is $\sum_{t \leq T} p^t \psi^t$.

As already mentioned, it takes a nontrivial amount of work to adapt the MWU to the NLP and MINLP settings: how do we relate θ to a given (MI)NLP, and how do we compute ψ? This paper provides some answers to these questions, based on three very different applications.

1.2 The MWU approximation guarantee

We recall here the precise statement of the MWU approximation guarantee. Given $q, T, \eta, \omega, \psi, p$ as above, it can be shown that [2]:

$$E_{\mathsf{MWU}} \triangleq \sum_{t \leq T} \psi^t p^t \quad \leq \quad \min_{i \leq q} \left(\sum_{t \leq T} \psi_i^t + \eta \sum_{t \leq T} |\psi_i^t| \right) + \frac{\ln q}{\eta}. \tag{1.2}$$

We remark that Eq. (1.2) is actually an immediate consequence of the more general bound:

$$\forall i \leq q \qquad E_{\mathsf{MWU}} \quad \leq \quad \sum_{t \leq T} \psi_i^t + \eta \sum_{t \leq T} |\psi_i^t| + \frac{\ln q}{\eta},$$

and that different weight update rules yield slightly different bounds.

1.3 The MWU in mixed-integer nonlinear programming

The objective of this paper is to propose an adaptation of the MWU to optimization problems in the following very general Mixed-Integer Nonlinear Programming (MINLP) form:

$$\left.\begin{array}{rrcl} \min\limits_{x \in \mathbb{R}^n} & f(x) & & \\ \forall \ell \leq m & g_\ell(x) & \leq & 0 \\ \forall j \in Z & x_j & \in & \mathbb{Z}, \end{array}\right\} [P] \tag{1.3}$$

where $Z \subseteq \{1, \ldots, n\}$ is given. If $Z = \varnothing$, the problem is called a Nonlinear Programming (NLP) problem.

There are three main issues in abstracting Alg. 2 to Eq. (1.3).

1. As presented in Alg. 2, the MWU algorithm appears to make a single decision (the investor's) based on a sequence of random predictions, whereas Eq. (1.3) makes n decisions.
2. The objective function $f(x)$ is given, and will generally be different than the cumulative weighted cost average that the MWU attempts to minimize.
3. Eq. (1.3) is hard to solve also because of feasibility, not just optimality. Nothing in Alg. 2 is about feasibility.

We shall address all three issues by means of a particular reformulation of Eq. (1.3) which partitions the decisions into two vectors of decision variables. One of the two vectors is simply x, the decision variables in Eq. (1.3). The other vector θ encodes (directly or indirectly) the values of some problematic terms in Eq. (1.3). The values for θ are decided using the MWU algorithm, whereas those for x are decided by solving the reformulation (with θ fixed at the values assigned by the MWU) at each iteration of the. Naturally, the replacement of problematic terms by easier terms in function θ will be chosen so as to make the resulting reformulation easier to solve than Eq. (1.3). We remark that the reformulation is parametrized by θ.

This strategy addresses the second and third issues: the current solution found by solving the reformulation at each MWU iteration can be "plugged into" Eq. (1.3): this yields an objective function value for Eq. (1.3) at each iteration, as well as a measure of infeasibility of the current solution. The cost vector ψ can be defined in terms of the relative progress of the objective function value, as well as the infeasibilities.

Let us see how this strategy addresses the first issue. Specifically, the vector θ in Alg. 2 is a vector of "random predictions" rather than decisions. This strategy, however, requires θ to match the value of the terms they replace, or at least get sufficiently close. We achieve this by modifying Step 2 in Alg. 3 so that θ is *updated* at each iteration by a factor sampled randomly in $[0, \omega_i]$ (see Alg. 3). This will help θ be "decided" by the MWU rather than "randomly sampled". Notice that the relative approximation guarantee of Sect. 1.2 still holds, since it only depends on the weights update rule.

Algorithm 3 Multiplicative Weights Update

1: **while** termination condition is not met **do**
2: for each $i \leq q$, multiply θ_i by a factor sampled randomly from $[0, \omega_i]$
3: compute the cost vector ψ associated to θ
4: update ω using ψ as in Eq. (1.1)
5: **end while**

In the strategy that we will propose in Section 3, θ could have a different dimension with respect to ω and ψ. This is one of the issues for which adapting Alg. 3 to applications requires nontrivial work.

The rest of this paper is organized as follows. In Sect. 2, we propose a reformulation-based methodology in order to relate θ to Eq. (1.3), and discuss the application of this methodology to three real-world applications. We present the adaptation of the MWU for (MI)NLP in Sect. 3, and the computational results in Sect. 4.

2 Pointwise reformulations

We introduce the reformulation referred to in Sect. 1.3. Broadly speaking, we reformulate P (see Eq. (1.3)) by replacing r "problematic" terms (e.g., the nonconvex terms) with simpler terms parametrized by θ. This yields a simplified formulation R in the original decision variables x, which varies in function of θ. We shall then iteratively solve R with θ fixed to values determined by the MWU framework. We remark that, with this approach, our MWU-based algorithm belongs to the category of *mat-heuristics* [31], meaning heuristics based on Mathematical Programming (MP).

Notationally, for a MP formulation P, we write $\mathsf{val}(P)$ to denote the objective function value of a global optimum of P, and $\mathsf{feas}(P)$ to denote the feasible set of P.

2.1 Definition
Given a MINLP P as in Eq. (1.3), a pointwise reformulation $R^\theta = \underset{\mathsf{t} \leftarrow \mathsf{t}'(\theta)}{\mathsf{ptw}}(P)$ is a family of MINLP formulations, parametrized by $\theta = (\theta_s \mid s \leq r)$, which are obtained by replacing given occurrences $\mathsf{t}_1, \ldots, \mathsf{t}_r$ of terms appearing in P by corresponding parametrized terms $\mathsf{t}'_s(\theta_s)$ (for $s \leq r$). □

Notationally, t and t' denote "terms" in the MINLP formulation P. We define a *term* formally as a symbolic expression represented by its parsing tree [15]. The parsing tree has leaf nodes labelled by the values of the constants or the index of the variables appearing in the term, and all other nodes labelled by the operators. The terms t' are expressed in function of the parameter vector θ, so we denote them $\mathsf{t}'(\theta)$. Both t and t' also depend on the decision variable vector x. These terms can be evaluated by means of a very simple recursive algorithm which starts at the root of the parsing tree: at each non-leaf

node v of the tree it calls itself on all the subnodes $v_1, \ldots, v_h$, and then returns the value $\otimes(v_1, \ldots, v_h)$ where $\otimes$ is the operator represented by the label of v; if v is a leaf node, the value of the constant or of the variable represented by the leaf node is returned. Through the algorithm, each term corresponds to a function $\mathbb{R}^n \to \mathbb{R}$, which we denote by $\mathsf{t}(x)$ and $\mathsf{t}'(\theta, x)$.

For every replaced term t_s (for $s \leq r$) in Defn. 2.1, let D_s be the interval range of $\mathsf{t}_s(x)$. For every replacement term t'_s (for $s \leq r$) let $D'_s(\theta_s)$ be the interval range of $\mathsf{t}'_s(\theta_s, x)$. For $s \leq r$, let Θ_s be the interval range of the corresponding parameter θ_s, and let $\Theta = (\Theta_s \mid s \leq r)$.

Given a parameter vector $\theta \in \Theta$ and a function ϕ, we denote by ϕ^θ the function obtained by replacing the terms t by the terms $\mathsf{t}'(\theta)$. Thus, for example, the objective and constraints of R are denoted as f^θ, g^θ, respectively.

We can therefore write a pointwise reformulation R^θ of P as follows:

$$\left.\begin{array}{lrcl} \min\limits_{x \in \mathbb{R}^{n'}} & f^\theta(x) & & \\ \forall \ell \leq m & g^\theta_\ell(x) & \leq & 0 \\ \forall j \in Z' & x_j & \in & \mathbb{Z}, \end{array}\right\} [R^\theta] \tag{2.1}$$

where $Z' \subseteq \{1, \ldots, n'\}$.

Note that Eq. (2.1) is actually a family of formulations, parametrized by θ. Note also that, whereas the number of variables n' may be different from n (since many variables occurring in replaced terms might be replaced terms involving fewer or more parameter symbols), P and R^θ have the same number of constraints m. If a replacement yields a trivial constraint, we stipulate that it is still formally part of Eq. (2.1). In practice, when solving pointwise reformulations, trivially satisfied constraints may be dropped, of course.

2.1 Theoretical properties

2.2 Definition
Given a MINLP P and $R^\theta = \underset{\mathsf{t} \leftarrow \mathsf{t}'(\theta)}{\mathsf{ptw}}(P)$, both defined on a vector x of decision variables in $\mathbb{R}^n$:

(a) R^θ is spanning if, for any $x \in \mathbb{R}^n$, there are values of θ such that evaluating the functions of P and of R^θ at x yields the same results — more precisely, $\exists \bar{\theta} \in \Theta_s$ such that

$$\forall s \leq r \qquad D_s \subset \bigcup_{\theta_s \in \Theta_s} D'_s(\theta_s) \quad \wedge \quad [\mathsf{t}'_s(\bar{\theta}_s)](x) = \mathsf{t}_s(x)$$

(note that the first condition in the conjunction is a form of consistency);

(b) R^θ is exact if, for each globally optimal solution x^* of P, there is at least one vector $\theta' \in \Theta$ such that x^* is also an optimal solution of $R^{\theta'}$;

(c) R^θ is efficient if there is a polynomial-time algorithm for approximately solving R^θ (for $\theta \in \Theta$) to within a given $\varepsilon > 0$ approximation factor. □

Note that the exactness property of a pointwise reformulation only makes sense for feasible problems. We arbitrarily define any pointwise reformulation of a class of infeasible problems to be exact.

More informally, we say that a pointwise reformulation is *good* if there is an established, practically efficient technology for solving R^θ either optimally or approximately. For example, if R^θ turns out to be a Linear Program (LP) or convex NLP (cNLP), then R^θ is efficient; if it turns out to be a Mixed Integer Linear Program (MILP), R^θ is good, since current MILP solution technology can routinely solve fairly large-scale MILPs, even though MILP itself is **NP**-hard [25]. Obviously, every efficient pointwise reformulation is also good.

2.3 Example
Consider the following formulation P:

$$\min\ x^2(1-y)+y \tag{2.2}$$
$$x+2y \geq 2 \tag{2.3}$$
$$y \in \{0,1\}. \tag{2.4}$$

If we set $y=0$, Eq. (2.3) and the objective function direction force $x=2$, whereas if $y=1$ we can let $x=0$; therefore the global optimum is $(x^*,y^*)=(0,1)$. We replace the term x^2 in the objective function by the term consisting of the scalar parameter θ, obtaining a pointwise reformulation R^θ:

$$\begin{aligned}\min\ (1-\theta)y+\theta & \\ x+2y &\geq 2\\ y &\in \{0,1\}.\end{aligned}$$

It is easy to see that R^θ is spanning whenever $\theta \in \mathbb{R}_+$. If we set $y=0$ we obtain $x \geq 2$, whereas $y=1$ yields no constraints on x. The objective function value if $y=0$ is θ; $y=1$ yields $1-\theta+\theta=1$. So for $\theta<1$ the set of global optima is $[2,\infty)\times\{0\}$; $\theta>1$ yields the global optimal set $\mathbb{R}_+\times\{1\}$, and if $\theta=1$ every feasible solution is optimal, with optimal objective function value equal to 1. Hence $(x,y)=(x^*,y^*)=(0,1)$ yields an optimum as long as $\theta \geq 1$, which means that this pointwise reformulation is exact. This pointwise reformulation is not efficient, since it is a MILP, but it is good. □

2.4 Lemma
Given P and a spanning reformulation $R^\theta = \underset{\mathsf{t}\leftarrow\mathsf{t}'}{\mathsf{ptw}}(P)$, we have:

$$\mathsf{feas}(P) \subseteq \bigcup_{\theta\in\Theta} \mathsf{feas}(R^\theta). \tag{2.5}$$

Proof. Let $x' \in \mathsf{feas}(P)$. Since R^θ is spanning, there is $\xi \in \Theta$ such that $\mathsf{t}_s(x') = [\mathsf{t}_s(\xi_s)](x')$ for each $s \leq r$, which implies $g_\ell^\xi(x^*) = g_\ell(x^*)$ for all $\ell \leq m$. Since $x' \in \mathsf{feas}(P)$, $g_\ell(x') \leq 0$ for all $\ell \leq m$, hence x' is also feasible in R^ξ. Since $\mathsf{feas}(R^\xi)$ is a subset of the right hand side of Eq. (2.5) for each possible $\xi \in \Theta$, the result follows. □

The following example shows that Eq. (2.5) cannot be tightened to an equality.

2.5 Example
Consider the pure feasibility NLP formulation F:

$$x \geq \frac{1}{2} \tag{2.6}$$
$$x^2 = x, \tag{2.7}$$

where x is a continuous decision variable. The constraint in Eq. (2.7) is equivalent to $x \in \{0,1\}$, so Eq. (2.6) forces $x=1$, hence $\mathsf{feas}(F)=\{1\}$. We rewrite $x^2 = x\,x$ and replace the first occurrence of x by θ, which yields the pointwise reformulation $\underset{\mathsf{t}\leftarrow\mathsf{t}'}{\mathsf{ptw}}(F) = R^\theta$:

$$x \geq \frac{1}{2} \tag{2.8}$$
$$\theta\, x = x. \tag{2.9}$$

Any value of $\theta \neq 1$ requires $x=0$ in order for Eq. (2.9) to hold, but $x=0$ is infeasible by Eq. (2.8). Setting $\theta=1$ yields $\mathsf{feas}(R^1) = [\frac{1}{2},\infty)$. In particular, we have

$$\{1\} = \mathsf{feas}(F) \subsetneq \bigcup_{\theta\in\mathbb{R}} \mathsf{feas}(R^\theta) = \mathsf{feas}(R^1) = [\tfrac{1}{2},\infty). \tag{2.10}$$

Since F and R^θ have no objective function, every feasible solution is optimal by definition. Verifying exactness reduces to checking that, for every feasible solution of F, there are values of θ such that the same solution is feasible in R^θ, which is established by Eq. (2.10). So this reformulation is exact. Since F is a nonconvex NLP but R^θ is an LP, this reformulation is efficient. □

A relaxation of a MP formulation Q provides a guaranteed bound (in the optimization direction) at every feasible point of Q. Since relaxations must be efficiently solvable, and since one usually looks for a bound to the *optimal* objective function value of Q, rather than to any objective function value achieved by a feasible point in Q, it makes sense to generalize a relaxation so it is about optimal rather than feasible points. A *bounding reformulation* of Q as a reformulation which, when solved to optimality, provides a bound in the optimization direction to the optimal objective function value of Q (and, moreover, its feasible set contains the feasible set of Q). Obviously, all relaxations are bounding reformulations.

2.6 Lemma
For any formulation P and spanning pointwise reformulation R^θ, there exists $\xi \in \Theta$ such that R^ξ is a bounding reformulation of P.

Proof. The proof of Lemma 2.4 implies that, if $\xi \in \Theta$ is such that $\mathsf{t}_s(x^*) = [\mathsf{t}_s(\xi_s)](x^*)$ for all $s \leq r$, $x^* \in \mathsf{feas}(R^\xi)$. Similarly, we show $f^\xi(x^*) = f(x^*)$, and therefore:

$$\mathsf{val}(R^\xi) \leq f^\xi(x^*) = f(x^*) = \mathsf{val}(P),$$

which establishes the result. □

We remark that the bounding reformulation guaranteed by Lemma 2.6 need not be a relaxation in the traditional sense.

2.7 Remark
Often, the replacement process $\mathsf{t} \leftarrow \mathsf{t}'(\theta)$ raises a cardinality issue: we replace r terms of the original formulation, and we have to use its solution to compute q costs, where r might in general be different from q. We shall discuss this issue in Sect. 3.1 below. □

Last but not least, note that pointwise reformulations can be used in more general settings than just the MWU algorithm. Although they have been devised with the MWU in mind, what they really achieve is a general mechanism for automatically decomposing the solution process of a MINLP in two phases: one for deciding values of θ, and the other for deciding values of x, based on solving the corresponding pointwise reformulation.

2.2 Applications

In this section, we present pointwise reformulations for three real-world MINLP problems, known to be difficult to solve both from a theoretical and a practical viewpoint, namely:

- the Distance Geometry Problem (DGP) with Euclidean distances;
- the Hydro-power (short-term) Unit Commitment problem (HUC);
- a subclass of nonconvex variants of the Markowitz' Mean-Variance Portfolio Selection Problem (MVPS).

The structure of each of those problems is exploited to construct a pointwise reformulation, in view of solving the problem using the MWU algorithm.

2.2.1 Distance Geometry Problem

The Distance Geometry Problem (DGP) with Euclidean distances is defined formally as follows [29]: given an integer $K > 0$, a simple undirected graph $G = (V, E)$, and an edge weight function $d : E \to \mathbb{R}_+$, establish or deny the existence of a vertex realization function $x : V \to \mathbb{R}^K$ such that:

$$\forall \{u, v\} \in E \qquad \|x_u - x_v\|_2 = d_{uv}. \tag{2.11}$$

The DGP arises in many important applications: determination of protein conformation from distance data [30], localization of mobile sensors in communication networks [18], synchronization of clocks from phase information [38], control of unmanned submarine fleets [3], spatial logic [19], and more [29]. It is **NP**-complete when $K = 1$ and **NP**-hard for larger values of K [34]. Notationwise, we let $n = |V|$ and $m = |E|$.

The most common MP formulation for the DGP is:

$$\min_x \sum_{\{u,v\} \in E} (\|x_u - x_v\|_2^2 - d_{uv}^2)^2, \tag{2.12}$$

It is obvious that the given DGP instance is YES if and only if the globally optimal value of Eq. (2.12) is zero. Note, however, that the DGP is not currently known to be in **NP** for $K > 1$ [6]. Therefore, if we wish to employ floating point based solution methods, we will not be able to determine precisely whether a given objective function value is exactly zero. We therefore take a more practical standpoint, and postulate that the quality of a solution is proportional to the value of a given error function.

Consider now the following formulation:

$$\left.\begin{array}{rcl} \max\limits_x & \sum\limits_{\{u,v\} \in E} \|x_u - x_v\|_2^2 & \\ \forall \{u, v\} \in E & \|x_u - x_v\|_2^2 & \leq \quad d_{uv}^2. \end{array}\right\} \tag{2.13}$$

2.8 Proposition
Eq. (2.13) *is an exact reformulation*[1] *of Eq.* (2.12).

Proof. Replacing the outer square in Eq. (2.12) by an absolute value trivially yields an exact reformulation:

$$\min_x \sum_{\{u,v\} \in E} |\|x_u - x_v\|_2^2 - d_{uv}^2|, \tag{2.14}$$

since Eq. (2.14) is zero iff Eq. (2.12) is zero. We reformulate Eq. (2.14) as:

$$\left.\begin{array}{rcl} \min\limits_{x, \hat{t} \geq 0} & \sum\limits_{\{u,v\} \in E} \hat{t}_{uv} & \\ \forall \{u, v\} \in E & -\hat{t}_{uv} \leq \|x_u - x_v\|_2^2 - d_{uv}^2 & \leq \quad \hat{t}_{uv}. \end{array}\right\} \tag{2.15}$$

The concave constraints $-\hat{t}_{uv} \leq \|x_u - x_v\|_2^2 - d_{uv}^2$ can be moved back to the objective, yielding:

$$\left.\begin{array}{rcl} \min\limits_{x, \hat{t} \geq 0} & \sum\limits_{\{u,v\} \in E} ((d_{uv}^2 - \|x_u - x_v\|_2^2) & + \quad \hat{t}_{uv}) \\ \forall \{u, v\} \in E & \|x_u - x_v\|_2^2 - d_{uv}^2 & \leq \quad \hat{t}_{uv}. \end{array}\right\} \tag{2.16}$$

Assume first that the given DGP is YES; then Eq. (2.15) must have globally optimal objective function value zero, which implies that $\hat{t}_{uv} = 0$ (for all $\{u, v\} \in E$) at the optimum, and so the following

[1] An *exact reformulation* (formally defined in [28] as a surjective mapping from the optima of the exact reformulation to the optima of the original problem) is not the same as an *exact pointwise reformulation*. Intuitively speaking, solving an exact reformulation of a problem directly yields a solution of the problem itself, which is not generally the case for an exact pointwise reformulation.

reformulation of Eq. (2.16),

$$\left.\begin{array}{rll} \min\limits_{x} & \sum\limits_{\{u,v\}\in E} (d_{uv}^2 - \|x_u - x_v\|_2^2) & \\ \forall \{u,v\} \in E & \|x_u - x_v\|_2^2 - d_{uv}^2 & \leq \quad 0, \end{array}\right\} \tag{2.17}$$

preserves a mapping between solutions of the DGP and global optima of Eq. (2.15). If the DGP is NO then the globally optimal objective function value of Eq. (2.15) must be strictly greater than zero, which implies that either Eq. (2.17) is infeasible, or it has a strictly positive globally optimal objective function value, whence Eq. (2.17) is an exact reformulation of Eq. (2.12). Finally, we eliminate the objective constant (which does not change global optima) and write $\min -f$ as $\max f$, obtaining the exact reformulation:

$$\left.\begin{array}{rll} \max\limits_{x} & \sum\limits_{\{u,v\}\in E} \|x_u - x_v\|_2^2 & \\ \forall \{u,v\} \in E & \|x_u - x_v\|_2^2 & \leq \quad d_{uv}^2, \end{array}\right\}$$

as claimed. □

2.9 Corollary
If the given DGP instance is YES, at any global optimum x^* of Eq. (2.13) all constraints ($\forall \{u,v\} \in E \; \|x_u - x_v\|_2^2 \leq d_{uv}^2$) of Eq. (2.13) are active.

Proof. By Lemma 2.8, Eq. (2.13) correctly solves a DGP instance where the given partial Euclidean Distance Matrix (pEDM) is (d_{uv}). The global optimum x^* must therefore satisfy Eq. (2.11), which implies that $\|x_u^* - x_v^*\|_2 = d_{uv}$ for each $\{u,v\} \in E$, as claimed. □

We can easily derive a pointwise reformulation of Eq. (2.13) by replacing the quadratic term $(x_{uk} - x_{vk})^2 = (x_{uk} - x_{vk})(x_{uk} - x_{vk})$ occurring in the objective function with a linear term $\theta_{uvk}(x_{uk} - x_{vk})$:

$$\left.\begin{array}{rll} \max\limits_{x} & \sum\limits_{\{u,v\}\in E} \sum\limits_{k\leq K} \theta_{uvk}(x_{uk} - x_{vk}) & \\ \forall \{u,v\} \in E & \|x_u - x_v\|_2^2 & \leq \quad d_{uv}^2. \end{array}\right\} \tag{2.18}$$

It is easy to see that Eq. (2.18) is spanning, since, given a solution x', it suffices to set $\theta'_{uvk} = (x'_{uk} - x'_{vk})$ for the objective and constraints of the pointwise reformulation to take identical values to the objective and constraints of the original formulation.

2.10 Proposition
If the given DGP instance is YES, for each globally optimal solution x^* of Eq. (2.13) there is a parameter matrix $\theta \in \mathbb{R}^{mK}$ such that x^* is a globally optimal solution of Eq. (2.18).

Proof. By Lemma 2.6, Eq. (2.18) is a bounding reformulation of Eq. (2.13) for $\theta^* = (x_{uk}^* - x_{vk}^* \mid \{u,v\} \in E \wedge k \in K)$, where x^* is a global optimum of Eq. (2.13). Since Eq. (2.18) is a maximization problem, we have to show that there is no optimum x' of Eq. (2.18) such that $f^{\theta^*}(x') > f^{\theta^*}(x^*)$. To aim at a contradiction, we suppose the existence of such an x'. Then there must be at least one edge $\{u,v\} \in E$ such that

$$\begin{array}{rrll} & \sum\limits_{k\leq K} \theta_{uvk}^*(x'_{uk} - x'_{vk}) & > & \|x_u^* - x_v^*\|_2^2 \\ \Rightarrow & \sum\limits_{k\leq K} (x_{uk}^* - x_{vk}^*)(x'_{uk} - x'_{vk}) & > & \|x_u^* - x_v^*\|_2^2 \\ \Rightarrow & (x_u^* - x_v^*)(x'_u - x'_v) & > & \|x_u^* - x_v^*\|_2^2. \end{array}$$

The last equation can be re-written as $y \cdot z > \|z\|^2$, for $y = x'_u - x'_v$ and $z = x_u^* - x_v^*$, and hence as $\|y\|\|z\|\cos(\mu) > \|z\|^2$, where μ is the angle between y and z. Since $\|z\| = 0$ yields $0 > 0$, we assume

$\|z\| > 0$, and we can hence divide through by $\|z\|$, yielding $\|y\| \cos(\mu) > \|z\|$. Since $\cos(\mu) \leq 1$, the only way the latter equation can hold is if $\|y\| > \|z\|$. Hence:

$$\begin{array}{rrcl} & \|x'_u - x'_v\|_2 & > & \|x^*_u - x^*_v\|_2 \\ \Rightarrow & \|x'_u - x'_v\|_2^2 & > & \|x^*_u - x^*_v\|_2^2 = d_{uv}^2 \end{array}$$

by Cor. 2.9. So $\|x'_u - x'_v\|_2^2 > d_{uv}^2$ which means that x' is infeasible in Eq. (2.18). The result follows. □

2.11 Theorem
The formulation in Eq. (2.18) *is an exact and efficient pointwise reformulation of Eq.* (2.11).

Proof. The fact that Eq. (2.18) is a pointwise reformulation of Eq. (2.11) follows by Prop. 2.8. The fact that it is exact follows by Prop. 2.10. The fact that it is efficient follows by the existence of polynomial-time algorithms for cNLP (see e.g. [12]). □

2.2.2 Hydro-power unit commitment

The short-term HUC problem is essentially a scheduling problem, defined as follows: over a uniformly discretized time horizon $H = \{1, \cdots, \bar{h}\}$ (ranging from one day to a week), given an initial water volume V_0 in a water reservoir, the goal is to find the optimal schedule of released water flow $(x_h \mid h \in H)$ (expressed in m^3) which maximizes the revenue obtained by providing the generated power $(y_h \mid h \in H)$ (expressed in MW) to the grid, such that the final volume of water remaining in the reservoir reaches a desired target $V_{\bar{h}}$ (again expressed in m^3).

Additional features of the model include:

- time period duration τ (expressed in h)
- volume bounds $\underline{V}$ and $\bar{V}$ (expressed in m^3) on the volume v_h, for each $h \in H$
- maximum flow bound $\bar{Q}$ (expressed in m^3/s) on the released flow x_h, for each $h \in H$
- maximum ramp-down Q^- and ramp-up Q^+ (expressed in $m^3/s/h$) for the flow x_h, for each $h \in H$
- forecasted inflows I_h (expressed in m^3/s), for each $h \in H$
- forecasted power selling prices Π_h (expressed in currency/MWh), for each $h \in H$
- parameters and coefficients $K_1, \ldots, K_6$, $L_1, \ldots, L_6$, $\underline{L}$ and R_0 of a polynomial function which models the generated power y_h, for each $h \in H$.

The simplified version can be formulated as follows, where v is the wolume of water in the reservoir:

$$\left.\begin{array}{rrcl} \max\limits_{x,y,v} & \sum\limits_{h \in H} \tau \Pi_h y_h & & \\ & v_0 & = & V_0 \\ & v_{\bar{h}} & = & V_{\bar{h}} \\ \forall h \in H & v_{h+1} - v_h & = & 3600\tau(I_h - x_h) \\ \forall h \in H & x_h - x_{h+1} & \leq & Q^- \\ \forall h \in H & x_{h+1} - x_h & \leq & Q^+ \\ \forall h \in H & y_h & = & \varphi(x_h, v_h) \\ \forall h \in H & v_h & \in & [\underline{V}, \bar{V}] \\ \forall h \in H & x_h & \in & [0, \bar{Q}], \end{array}\right\} \tag{2.19}$$

where $\varphi(x, v) = 9.81x(\sum\limits_{l=0}^{6} L_l x^l)(\sum\limits_{k=0}^{6} K_k v^k - \underline{L} - R_0 x^2)$ is the function expressing the power generated depending on the water flow released x and the water volume v in the reservoir.

By replacing each nonconvex multivariate function $\varphi(x_h, v_h)$ (for $h \in H$) with an affine approximation (i.e. a first-order approximation at a given point $(\tilde{x}_h, \tilde{v}_h)$), we derive a pointwise linear reformulation of Eq. (2.19):

$$\left.\begin{array}{rrcl} \max\limits_{x,y,v} & \sum\limits_{h\in H} \tau\Pi_h y_h & & \\ & v_0 & = & V_0 \\ & v_{\bar{h}} & = & V_{\bar{h}} \\ \forall h \in H & v_{h+1} - v_h & = & 3600\tau(I_h - x_h) \\ \forall h \in H & x_h - x_{h+1} & \leq & Q^- \\ \forall h \in H & x_{h+1} - x_h & \leq & Q^+ \\ \forall h \in H & y_h & = & \nu_{0h} + \nu_{1h}(x_h - \tilde{x}_h) + \nu_{2h}(v_h - \tilde{v}_h) \\ \forall h \in H & v_h & \in & [\underline{V}, \bar{V}] \\ \forall h \in H & x_h & \in & [0, \bar{Q}], \end{array}\right\} \tag{2.20}$$

We remark that, in the pointwise reformulation Eq. (2.20), the parameter vector θ is structured as the $h \times 5$ matrix $(\tilde{x}, \tilde{v}, \nu_0, \nu_1, \nu_2)$.

2.12 Lemma
The pointwise reformulation Eq. (2.20) *is spanning.*

Proof. For each $h \in H$, the following holds: if $x'_h \in [0, \bar{Q}]$ and $v'_h \in [\underline{V}, \bar{V}]$, the replacement term $\nu_{0h} + \nu_{1h}(x_h - \tilde{x}_h) + \nu_{2h}(v_h - \tilde{v}_h)$ matches the replaced term $\varphi(x_h, v_h)$ for all values of θ satisfying $\tilde{x}_h = x'_h$, $\tilde{v}_h = v'_h$ and $\nu_{0h} = \varphi(x'_h, v'_h)$. □

2.13 Proposition
For each globally optimal solution $X^* = (x^*, v^*, y^*)$ of Eq. (2.19)*, there is a $\theta^* \in \mathbb{R}^{5\bar{h}}$ such that X^* is a globally optimal solution of Eq.* (2.20)*.*

Proof. Let us show that X^* is a globally optimal solution of the pointwise problem Eq. (2.20) for

$$\theta^*_h = (x^*_h, v^*_h, \varphi(x^*_h, v^*_h), \frac{\partial\varphi}{\partial x_h}(x^*_h, v^*_h), \frac{\partial\varphi}{\partial v_h}(x^*_h, v^*_h)),$$

where $h \in H$. Since this definition of θ^* satisfies the spanning property in Prop. 2.12, we can invoke Lemma 2.4 to conclude that X^* is feasible for Eq. (2.20). Since X^* is a global optimum of Eq. (2.19), in particular it is also a local optimum, and therefore it satisfies the Karush-Kuhn-Tucker (KKT) conditions. By the choice of θ^*, the gradients of the objective function and the constraints of Eq. (2.20) at X^* are identical to gradients of objective and constraints of Eq. (2.19) at X^*. Therefore, X^* also satisfies first-order optimality conditions of Eq. (2.20) when parametrized by θ^*. Since Eq. (2.20) is an LP, any KKT point is also a global optimum, which concludes the proof. □

2.14 Theorem
The formulation in Eq. (2.20) *is an exact and efficient pointwise reformulation of Eq.* (2.19)*.*

Proof. Exactness of Eq. (2.20) follows by Prop. 2.13. Efficiency follows because Eq. (2.20) is an LP, which can be solved in polynomial time. □

2.2.3 Mean-variance portfolio selection

Our third application of pointwise reformulations is an uncountable class of problems, parametrized by symbols which denote abstract univariate functions (any assignment of concrete functions to the symbols

represents a different problem). Using this very general setting, we mean to provide an insight of how much better or worse the MWU performs, compared to MS, depending on how far or close such functions are to being linear. In Sect. 4 below, we shall show that the MWU performs better than the MS on functions which are "very nonlinear", whereas the MS wins out on functions which are "close to linear". Since this is only a qualitative statement derived empirically on a few univariate functions, we leave the interpretation of these informally expressed notions to the inspection of the function graphs (see Fig. 4).

The MVPS problem [32] is defined as follows: given a set of n possibly risky stocks, characterized by a mean return vector $\rho \in \mathbb{R}^n$ and a covariance matrix $Q \in \mathbb{R}^{n \times n}$, determine a composition of the portfolio, i.e., the fraction x_i of value invested in stock $i \leq n$, by simultaneously considering two conflicting targets: maximizing expected return of the portfolio and minimizing a measure of its risk. For an in-depth analysis on mean-variance approaches to portfolio selection problems, we refer the reader to the survey [17].

We consider here the MVPS variant with transaction costs and maximum number of held assets (also called the *sparsity* of the portfolio). Transaction costs are usually defined by means of a separable function $C : \mathbb{R}^n \to \mathbb{R}_+$ of the assets held in the portfolio:

$$C(x) = \sum_{i=1}^{n} C_i(x_i), \tag{2.21}$$

where, for all $i \leq n$, $C_i : \mathbb{R} \to \mathbb{R}_+$ is a univariate possibly nonconvex and nonconcave function. Note that this setting is much more general than the one usually considered in literature, involving concave non-decreasing transaction cost (see [26, 41]).

Let $K \in \mathbb{R}_+$ and $\sigma \in \mathbb{R}_+$ be respectively the maximum sparsity level and the maximum risk desired for the portfolio. The MVPS variant we are interested in can be formally stated as follows:

$$\left.\begin{array}{rcl} \max\limits_{x} \quad \rho^\top x & - & C(x) \\ \mathbf{1}\, x & = & 1 \\ x^\top Q x & \leq & \sigma \\ \|x\|_0 & \leq & K \\ x & \in & [\underline{x}, \bar{x}] \cup \{0\}, \end{array}\right\} \tag{2.22}$$

where $\mathbf{1} \in \mathbb{R}^n$ is the all-one n-dimensional vector, $\|x\|_0$ is the so-called ℓ_0-norm of x, i.e., $\|x\|_0 = |\{x_i : x_i \neq 0\}|$, $\underline{x}$ and $\bar{x}$ are respectively minimum and maximum buy-in thresholds, and x is a vector of n semi-continuous decision variables [21, 39].

In Eq. (2.22), the first constraint ensures the whole available capital is invested in the portfolio. Portfolio sparsity is used for modeling the will of investors to limit e.g. brokerage fees, bid-ask spreads and monitoring costs [35]. Moreover, we assume $0 \leq \underline{x} \leq \bar{x}$, so that short selling, i.e., the possibility for the investor to sell financial assets he/she does not own, is avoided. Eq. (2.22) is an **NP**-hard problem, even if $n = 3$ [8, 37], and can be exactly reformulated (in the sense of [28]) as the following non-convex MINLP:

$$\left.\begin{array}{rcl} \max\limits_{x,y} \quad \rho^\top x & - & C(x) \\ \mathbf{1}\, x & = & 1 \\ x^\top Q x & \leq & \sigma \\ \mathbf{1}\, y & \leq & K \\ y^\top \underline{x} & \leq x \leq & y^\top \bar{x} \\ y & \in & \{0,1\}^n. \end{array}\right\} \tag{2.23}$$

We can obtain a pointwise reformulation by replacing terms $C_i(x_i)$ with terms $(1 + x_i)\theta_i$, for every $i \leq n$. This yields the pointwise reformulated objective function:

$$\rho^\top x - \sum_{i \leq n} (\theta_i x_i + \theta_i) = (\rho - \theta)^\top x - \mathbf{1}\, \theta,$$

which is an affine function of x. Thus, the pointwise reformulation is:

$$\left.\begin{array}{rrcl} \max\limits_{x,y} & (\rho-\theta)^\top x & - & \mathbf{1}\,\theta \\ & \mathbf{1}\,x & = & 1 \\ & x^\top Q x & \leq & \sigma \\ & \mathbf{1}\,y & \leq & K \\ & y^\top \underline{x} \leq & x & \leq y^\top \bar{x} \\ & y & \in & \{0,1\}^n. \end{array}\right\} \tag{2.24}$$

2.15 Remark
The pointwise reformulation in Eq. (2.24) *is spanning, since the replacement terms* θ_i *match the replaced terms* $\frac{C_i(x_i)}{x_i+1}$ *at each feasible point* $(x_i \mid i \leq n)$*: it suffices to divide* $C_i(x_i) = (1+x_i)\theta_i$ *through by* $1+x_i$*. By Lemma 2.6, there exist values of* θ *which make Eq.* (2.24) *a bounding reformulation for the original problem Eq.* (2.22). □

2.16 Example
The pointwise reformulation in Eq. (2.24) *is* not *exact. It suffices to consider a portfolio with* $n = 2$ *stocks,* $Q = I_2$*,* $K = 2$*,* $\underline{x} = 0$*,* $\bar{x} = 0.55$*, i.e. essentially unconstrained apart from the total budget constraint* $x_1 + x_2 = 1$*, with* $\rho = (1, 0.40)^\top$*, and* $C(x) = (C_1(x_1), C_2(x_2))^\top = (x_1, 0)^\top$*. Note that, for all* $i \in \{1, 2\}$*,* $\theta_i = \frac{C_i(x_i)}{1+x_i}$*, and hence* $\Theta_i = [C_i(\underline{x})/(1+\bar{x}), C_i(\bar{x})/(1+\underline{x})]$*. Therefore, in this example,* $\Theta_1 = [0, 0.55]$ *and* $\Theta_2 = \{0\}$*. The objective function of the original formulation Eq.* (2.22) *is simply* $\max(0.40x_2)$*, which implies a global optimum* $x^* = (0.45, 0.55)^\top$*. The objective function of the pointwise reformulation is* $\max((1-\theta_1)x_1 + 0.40x_2 - \theta_1)$*. Now,* $x^* = (0.45, 0.55)^\top$ *is a global optimum of the pointwise reformulation if and oly if* $(1-\theta_1) < 0.40$*, i.e., iff* $\theta_1 > 0.60$*: however, these values of* θ_1 *do not belong to the set* Θ_1*.*

2.17 Remark
The pointwise reformulation in Eq. (2.24) *is not efficient, since the MVPS is* **NP***-hard [8]. However, the pointwise reformulation in Eq.* (2.24) *is good, since the solver technology for finding solutions of convex MIQPs is more advanced and reliable than that for solving nonconvex nonconcave MINLP.*

□

The previous models can be easily extended in order to take into account the presence of fixed transaction costs. Furthermore, the simplicity of the previous proposed approach makes it a promising pointwise reformulation candidate for the extension to other mixed integer nonlinear problems with separable nonconvexities.

3 The MWU for MINLPs

The MWU (Alg. 3) samples some values from an iteratively updated distribution in order to optimize a given loss or gain criterion. We adapt this framework to (MI)NLP by exploiting good pointwise reformulations, which can be solved more efficiently than the original formulation, and hence can be solved repeatedly at a relatively low computational cost. We rely on a set of parameters θ which can be guessed using the MWU framework. This yields a loop which updates the values of θ, uses them to solve a pointwise reformulation, and then estimates the error over each variable in order to update θ again at the next iteration.

The pseudocode of the MWU algorithm for MINLP is shown in Alg. 4. It takes a MINLP formulation P and a pointwise reformulation $\underset{\mathsf{t}\leftarrow\mathsf{t}'(\theta)}{\mathsf{ptw}}(P)$ as inputs, and hopefully produces a good solution as output.
Step 5 in Alg. 4 is not actually needed to ensure that the relative approximation guarantee Eq. (1.2) holds. Without it, however, the MWU is computationally much less efficient and effective.

Algorithm 4 MWU(P)

1: Initially set the iteration index $t = 1$, weights $\omega^t = \mathbf{1}$, parameters θ^{t-1} chosen uniformly at random in Θ, an incumbent $x^* = \infty$ and $t = 1$
2: **while** $t \leq T$ **do**
3: assign $\theta^t \leftarrow \theta^{t-1}\bar{\omega}^t$ where $\bar{\omega}^t$ is vector chosen uniformly at random in $[0, \omega^t]$
4: solve $\underset{\mathsf{t}\leftarrow\mathsf{t}'(\theta^t)}{\mathsf{ptw}}(P)$, get solution x^t
5: refine x^t (e.g. using local descent on P)
6: if x^t is better than the incumbent, replace $x^* \leftarrow x^t$
7: compute costs $\psi^t \in [-1, 1]^q$ from x^t
8: update weights for the next iteration: $\forall i \leq q\ \omega_i^{t+1} \leftarrow \omega_i^t(1 - \frac{\psi_i^t}{2})$
9: normalize weights such that $\sum_{i\leq q} \omega_i^{t+1} = 1$
10: increase t
11: **end while**

We now clarify the dependencies between symbols occurring in the MWU algorithm, symbols occurring in the pointwise reformulation, and the optima of the original formulation. E_{MWU}, the function that the MWU aims at minimizing, depends on p and ψ; p depends on ω, which is updated using ψ, and ψ depends on the local solution x of the pointwise reformulation, which replaces the incumbent x^* whenever it improves it (with respect to the objective function of the original formulation). Note also that x depends on θ through the pointwise reformulation, and that θ is randomly chosen from the discrete distribution p, proportional to ω.

Below, we discuss Steps 3, 4-5 and 7 of Alg. 4 in more detail.

3.1 Sampling

As mentioned in Remark 2.7, the dimension r of θ and the dimension q of ω and ψ might be different. The question is how to apply Step 2 in Alg. 3, since it implicitly assumes that $r = q$. We deal with this issue by defining θ using aggregations (if $r > q$) or disaggregations (if $q > r$) of the values $\bar{\omega}$ sampled from $[0, \omega]$ (see Step 3 in Alg. 4). Aggregations and disaggregations are obtained by applying any number of operators, such as products or sums, to $\bar{\omega}$. Since there are many ways to split products and sums in a prescribed number of different parts, the precise details of this step are heuristic in nature. We note that the MWU performance guarantee on E_{MWU} (see Eq. (1.2)) depends on q but not on r, and so it is not impacted by such details.

In Sect. 3.3 we sketch a general methodology which eschews this issue by ensuring that $r = q$ (so that sampling θ from the distribution p poses no problem). This methodology is then applied to the HUC and MVPS problems. In the DGP application, on the other hand, we have $mK = r > q = n$, where $n = |V|$, $m = |E|$ and $(K, G = (V, E, d))$ is the DGP input.

3.2 Solution and refinement

3.2.1 Solving pointwise reformulations

If P has no integer variable and $\underset{\mathsf{t}\leftarrow\mathsf{t}'}{\mathsf{ptw}}(P)$ is efficient, it is likely to be an LP or a cNLP, both of which can be either solved or accurately approximated in polynomial time. If the pointwise reformulation is not efficient but at least good, it may be a type of nonconvex NLP for which we have a practically fast solver which scales reasonably well.

If P has some integer variables and $\underset{t \leftarrow t'}{\mathsf{ptw}}(P)$ is good, P might be a MILP or a convex MINLP (cMINLP). This complicates matters, since solving a MILP or a cMINLP to optimality at each iteration is usually computationally costly, even if good solver technologies exist. Note, however, that all the MWU algorithm needs in order for its guarantee to hold is simply an error vector ψ^t at each iteration $t \leq T$ and the weight update rule Eq. (1.1). So in fact we can run *any heuristic we like* on the pointwise reformulation. In practice, we run Branch-and-Bound methods with a bound on computation time or optimality gap.

3.2.2 Refining the pointwise optimum

The refinement step is optional in theory, but computational experience shows it is necessary in practice for the MWU to perform well. If x' is the solution of the pointwise reformulation, any solver which is designed to improve x' with respect to the *original* formulation P (at least locally) can be used to refine x'.

3.3 Computing the MWU costs

This is the most critical step of the MWU algorithm, since it influences the performance guarantee. It has two requirements:

(a) $\psi^t \in [-1, 1]^q$ for all $t \leq T$;
(b) for any $t \leq T$, ψ^t measures the error of the current local solution x^t of the pointwise reformulation with respect to optimality and feasibility of the original formulation.

We need (a) to prove the MWU relative approximation guarantee, and (b) in order to relate E_{MWU} to the solution quality of the incumbent x^* (see issues 2-3 in Sect. 1.3).

Based on (a) and (b), we compute a scalar α^t related to optimality, and a set $\{\beta^t_\ell \mid \ell \leq m\}$ of vectors related to feasibility. Since we penalize infeasibilities but we generally do not award "better feasibility", the components of β^t_ℓ are usually required to be in $[0, 1]$ rather than $[-1, 1]$.

More specifically, let

$$R^{\theta^t} = \underset{t \leftarrow t'(\theta^t)}{\mathsf{ptw}}(P)$$

be the pointwise reformulation at iteration $t \leq T$, let $f(x), g_\ell(x) \leq 0$ (for $\ell \leq m$) be the objective function and constraints of P as per Eq. (1.3), and let $f^{\theta^t}(x), g^{\theta^t}_\ell(x) \leq 0$ be those of R^t (for $\ell \leq m$). After Steps 4-5 of Alg. 4, we can evaluate the current solution x^t in the pointwise reformulation by computing $f^{\theta^t}(x^t)$ and $g^{\theta^t}_\ell(x^t)$ for each $\ell \leq m$. We define arrays of values, α^t, β^t at each iteration $t \leq T$:

- let α^t be proportional to $f^{\theta^t}(x^t) - f(x^t)$, so as to favor a pointwise reformulation with a lower objective value (for a MINLP in the general minimization form Eq. (1.3); if considering a maximization problem, α^t should be replaced by $-\alpha^t$);
- for all $\ell \leq m$, let β^t_ℓ be proportional to $\max(g^{\theta^t}_\ell(x^t), 0)$, so as to penalize a pointwise reformulation which makes a feasible solution infeasible.

The arrays α, β can be scaled in any way which makes them satisfy requirement (a) above. We assume this scaling is application-dependent. We can now define ψ^t in a very simple way as the concatenation of α^t and β^t_ℓ for all $\ell \leq m$, which also fixes $q = 1 + m$. Other application-dependent ways of defining ψ by means of α, β are also possible (see Sect. 3.4.1).

3.4 Adapting the MWU for MINLP to applications

In this section we discuss the adaptations of Alg. 4 to the different application settings we considered.

3.4.1 Distance Geometry Problem

In this section, we discuss the adaptation of Alg. 4 to the DGP application setting.

The pointwise reformulation Eq. (2.18) we employ for the DGP relies on $\theta^t = (\theta^t_{uvk} \mid \{u,v\} \in E \wedge k \leq K)$ having dimension $r = K|E|$, $x^t = (x^t_v \mid v \in V)$ having dimension $n = |V|$, and $\psi^t = (\psi^t_{uv} \mid \{u,v\} \in E)$ having dimension $q = |E|$, at any iteration $t \leq T$. Also, the original formulation Eq. (2.11) is a pure feasibility problem consisting of $|E|$ quadratic equations, without objective function or integrality constraints. We therefore define:

$$\forall \{u,v\} \in E \quad \psi^t_{uv} = \frac{|\,\|x^t_u - x^t_v\|_2 - d_{uv}|}{\max(\|x^t_u - x^t_v\|_2, d_{uv})}. \tag{3.1}$$

This definition of ψ is close to the interpretation of β given in Sect. 3.3. Since β^t_ℓ is defined for the ℓ-th inequality constraint, and equality constraints correspond to pairs of opposing inequality constraints, we simply observe that $\max(\tilde{g}_\ell(x^t), 0)$ turns into

$$\max(g^{\theta^t}_\ell(x^t), -g^{\theta^t}_\ell(x^t)) = |g^{\theta^t}_\ell(x^t)| = |\,\|x^t_u - x^t_v\|_2 - d_{uv}|,$$

which is the numerator in the definition of ψ^t in Eq. (3.1).

As mentioned above and in Sect. 3.1, in the DGP application we have a discrepancy between r and q; accordingly, we sample a random vector $\bar{\omega}^t$ uniformly at random from $[0, \omega^t]$, and define θ^{t+1} as a disaggregation of $\bar{\omega}^t$ over $k \in K$, as follows:

$$\forall k \leq K, \{u,v\} \in E \quad \theta^{t+1}_{uvk} = \bar{\omega}^t_{uv}(x^t_{uk} - x^t_{vk}),$$

i.e., making a reasonable guess, we postulate that a good parameter θ^{t+1}_{uvk} will be proportional to the k-th component of the vector $x^t_u - x^t_v$.

The performance guarantee of the MWU applied to the DGP turns out to be slightly better than the general one in Eq. (1.2), due to the fact that infeasibility is penalized but feasibility is not rewarded:

$$\min_{t \leq T} \sum_{\{u,v\} \in E} \psi^t_{uv} p^t_{uv} \leq \frac{1}{T}\left(\frac{\ln|E|}{\eta} + (1+\eta) \min_{\{u,v\} \in E} \sum_{t \leq T} \psi^t_{uv}\right), \tag{3.2}$$

which is actually a statement on a weighted feasibility error of the MWU solution x^*, since it concerns the minimum over all iterations. For a proof of this statement, see [16, Prop. 4.1].

3.4.2 Hydro-power unit commitment

In this section, we discuss the adaptation of Alg. 4 to the HUC application setting.

For each time period of the scheduling horizon, an upper bound on the contribution to the overall revenue is given by $\bar{\Pi}\bar{\varphi} \triangleq \max_h \Pi_h \max_{x,v} \varphi(x,v)$. This allows us to define costs:

$$\forall h \in H \quad \psi^t_h = \frac{\bar{\Pi}\bar{\varphi} - \Pi_h y_h}{\bar{\Pi}\bar{\varphi}}. \tag{3.3}$$

This definition of ψ^t is close to the interpretation of α^t given in Sect. 3.3. The overall gain/cost reward is scaled appropriately for each time period.

The pointwise reformulation we employ for the HUC (Eq. (2.20)) relies on a parameter matrix $\theta^t = (\tilde{x}^t, \tilde{v}^t, \nu_0^t, \nu_1^t, \nu_2^t)$, and $\bar{h}$-dimensional cost vectors ψ^t. Note that θ^t is $\bar{h}\times 5$, and that the first two vectors are actually points in the (x, v)-space. We therefore set $\tilde{x}^t = x^{t-1}$, $\tilde{v}^t = v^{t-1}$, and sample ν_a^t (for $a \in \{0, 1, 2\}$) from ω^t: it suffices to draw three different samples from the same distribution p proportional to ω^t. More precisely,

$$\begin{array}{rrcl} \forall h \in H, & \nu_{0h}^t & = & \bar{\omega}_h^t \varphi(\tilde{x}_h^t, \tilde{v}_h^t), \\ & \nu_{1h}^t & = & \bar{\omega}_h^t \frac{\partial \varphi}{\partial x_h}(\tilde{x}_h^t, \tilde{v}_h^t), \\ & \nu_{2h}^t & = & \bar{\omega}_h^t \frac{\partial \varphi}{\partial v_h}(\tilde{x}_h^t, \tilde{v}_h^t). \end{array}$$

Since the distance to $\bar{\Pi}\bar{\varphi}$ is always penalized with nonnegative ψ, the performance guarantee of the MWU applied to the HUC is similar to Eq. (3.2):

$$\min_{t\leq T} \sum_{h\in H} \psi_h^t p_h^t \leq \frac{1}{T}\left(\frac{\ln \bar{h}}{\eta} + (1+\eta) \min_{h\in H} \sum_{t\leq T} \psi_h^t\right). \tag{3.4}$$

3.4.3 Mean-variance portfolio selection

Note that the only complicating terms of the MVPS formulation Eq. (2.23) are the transaction costs in the objective function. Since feasibility is not an issue, we only need to define ψ for optimality purposes, i.e. we only consider the α arrays mentioned in Sect. 3.3. Specifically, at each iteration $t \leq T$, we define a vector $\alpha^t \in [0, 1]^n$, whose components are:

$$\alpha_i^t = \frac{C_i(x_i^t) - (x_i^t + 1)\theta_i^t}{\max(|(x_i^t + 1)\theta_i^t|, |C_i(x_i^t)|)}. \tag{3.5}$$

Note that we define α^t to be a vector of n components, instead of a scalar as in Sect. 3.3. This adaptation fits the MVPS well, since its objective function is separable, and since satisfying its constraints is not hard. Each component of α^t defines a cost/gain relative to the contribution of each asset quantity towards the value of the objective function. We then simply set:

$$\forall i \leq n \quad \psi_i^t = \alpha_i^t. \tag{3.6}$$

In Alg. 4, the θ^t vector is updated in terms of ω, which are themselves updated as in Eq. (1.1). For this application, we decided to add a step with the purpose of explicitly scale each θ_i^t (for $i \leq n$) by $\mathtt{t}_i = \frac{C_i(x_i^t)}{(1+x_i^t)}$ first, in order to make θ_i^t a better replacement candidate for $\mathtt{t}_i$:

$$\forall i \leq n \quad \theta_i^t = \psi_i^t \frac{C_i(x_i^t)}{(1 + x_i^t)}. \tag{3.7}$$

After this modification, we resume Alg. 4: we proceed to update ω, then compute θ^{t+1} as an update to θ^t multiplied by random sample from the distribution p, proportional to ω.

4 Computational experiments

In this section we present comparative computational experiments to validate the behaviour of the MWU algorithm for (MI)NLP in practice. We always compare the MWU with a classic MS heuristic, which is possibly the most similar existing algorithm to the MWU for (MI)NLP, on a fixed number T of iterations, and with initial points sampled uniformly within the instance ranges. We chose $\eta = 0.5$ and $T = 20$ for most of the experiments (unless stated otherwise).

4.1 DGP results

The test set consists of a set of 46 Protein Data Bank (PDB) [7] files, to be realized in $\mathbb{R}^3$. The protein graph has an edge $\{i, j\}$, weighted by the inter-atomic distance between atoms i and j, whenever this distance is smaller than 5Å (compatibly with Nuclear Magnetic Resonance [36] data). Note that, since all the proteins we tested actually exist, all of these instances are YES instances of the DGP. Based on the computational results, we partition these instances in two groups: Easy (27 instances) and Hard (19 instances). We configured both MWU and MS solvers with $T = 20$ iterations, and use the IPOpt solver [14] as the local NLP solver in both methods.

						MWU			MS		
Instance	n	m	d:min	max	avg	err:avg	max	CPU	avg	max	CPU
C0700odd.1	8	28	0.93	4.06	2.39	0.00	0.00	0.61	0.00	0.00	**0.46**
C0700odd.2	8	28	0.93	4.06	2.39	0.00	0.00	0.61	0.00	0.00	**0.46**
C0700odd.3	8	28	0.93	4.06	2.39	0.00	0.00	0.62	0.00	0.00	**0.46**
C0700odd.5	8	28	0.93	4.06	2.39	0.00	0.00	0.61	0.00	0.00	**0.39**
C0700odd.6	8	28	0.93	4.06	2.39	0.00	0.00	0.60	0.00	0.00	**0.49**
C0700odd.7	8	28	0.93	4.06	2.39	0.00	0.00	0.64	0.00	0.00	**0.39**
C0700odd.8	8	28	0.93	4.06	2.39	0.00	0.00	0.61	0.00	0.00	**0.45**
C0700odd.9	8	28	0.93	4.06	2.39	0.00	0.00	0.61	0.00	0.00	**0.39**
C0700odd.A	8	28	0.93	4.06	2.39	0.00	0.00	0.61	0.00	0.00	**0.45**
C0700odd.B	8	28	0.93	4.06	2.39	0.00	0.00	0.62	0.00	0.00	**0.44**
lavor11	11	40	1.48	4.87	2.77	0.00	0.00	0.77	0.00	0.00	**0.51**
lavor11_7-2	11	47	1.49	4.97	2.95	0.00	0.00	**0.72**	0.00	0.00	0.83
lavor11_7	11	47	1.44	4.98	2.96	0.00	0.00	**0.68**	0.00	0.00	0.81
lavor11_7-1	11	47	1.44	4.98	2.96	0.00	0.00	**0.73**	0.00	0.00	0.80
lavor11_7-b	11	47	1.44	4.98	2.96	0.00	0.00	**0.78**	0.00	0.00	0.87
C0150alter.1	26	191	1.02	4.99	2.99	0.00	0.00	**2.01**	0.00	0.00	5.24
lavor30_6-4	30	191	0.64	4.96	2.88	0.00	0.00	**1.74**	0.00	0.00	4.00
lavor30_6-1	30	192	0.64	4.98	2.90	0.00	0.00	**1.85**	0.00	0.00	3.29
lavor30_6-8	30	193	0.64	4.99	2.90	0.00	0.00	**1.91**	0.00	0.00	3.85
lavor30_6-6	30	195	0.64	4.99	2.96	0.01	0.15	**1.71**	0.00	0.00	3.47
lavor30_6-7	30	195	0.64	4.98	2.93	0.00	0.00	**1.87**	0.00	0.00	4.11
lavor30_6-3	30	195	0.64	4.98	2.92	0.00	0.00	**1.73**	0.00	0.00	3.65
lavor30_6-5	30	195	0.64	4.99	2.93	0.00	0.00	**1.79**	0.00	0.00	3.37
lavor30_6-2	30	202	0.64	4.99	3.03	0.00	0.00	**1.81**	0.00	0.00	4.45
tiny	27	252	0.93	4.99	3.10	0.00	0.00	**1.56**	0.00	0.00	8.75
C0700.odd.G	36	308	0.98	4.99	3.04	0.00	0.00	**2.58**	0.00	0.00	8.29
C0700.odd.H	36	308	0.98	4.99	3.04	0.00	0.00	**2.55**	0.00	0.00	10.01
avg (CPU)								**1.22**			2.62
geo (CPU)								**1.05**			1.40

Table 1: Comparative results on the Easy instances of the DGP. The last line reports the arithmetic and geometric means of the CPU times.

The result tables 1 and 2 report, for each of the instance sets Easy and Hard, the instance PDB name, the number n of atoms, the number m of given distances, the minimum, maximum and average distance values in the instance, and the solver performance statistics. These are average and maximum edge errors, defined respectively as:

$$\frac{1}{m} \sum_{\{i,j\} \in E} |d_{ij} - \|x_i - x_j\|_2 |$$

$$\max_{\{i,j\} \in E} |d_{ij} - \|x_i - x_j\|_2 |,$$

and seconds of user CPU time of an i7 at 2GHz with 8GB RAM running Darwin 13.4.0.

Both methods are iterative, both have been run for the same number of iterations, MWU calls the local NLP solver twice per iteration (solution of the pointwise reformulation and refinement) and MS calls it only once. However, the MWU takes much less time (Fig. 1), and yields better results, as shown in Tables 1-2.

For the Hard instances only, we also carried out a comparison to a very well-known and efficient metaheuristic called Variable Neighbourhood Search (VNS) [23], applied to the DGP as described in [27], and using IPOpt as its local NLP solver. The range of applications where VNS excels borders on unbelievable. As is clear from the results in Table 3 and Fig. 2, it excels in the DGP case too, in

						MWU			MS		
Instance	n	m	d:min	max	avg	err:avg	max	CPU	avg	max	CPU
2erl-frag-bp1	39	406	0.81	4.99	3.11	0.00	0.00	**2.63**	0.00	0.00	15.06
C0080create.1	60	681	0.98	5.00	3.33	**0.04**	**0.41**	**6.17**	0.26	1.27	37.79
C0080create.2	60	681	0.98	5.00	3.33	**0.04**	**0.41**	**6.05**	0.06	0.99	36.50
names	82	840	1.21	5.00	3.48	0.16	1.28	**10.91**	**0.15**	**1.17**	79.85
pept	107	999	1.16	5.00	3.55	**0.16**	**1.38**	**15.53**	0.42	1.45	108.80
C0020pdb	107	999	1.16	5.00	3.55	**0.16**	**1.45**	**14.69**	0.44	1.92	101.72
1guu-1	150	959	1.20	4.99	3.46	**0.09**	**0.79**	**15.09**	0.10	1.06	30.80
1guu-4000	150	968	0.33	4.99	3.51	**0.11**	**0.68**	**19.31**	0.16	1.00	42.88
1guu	150	955	1.30	5.00	3.46	**0.09**	**0.70**	**15.33**	0.12	1.07	31.58
res_5000	108	1392	0.94	5.00	3.42	**0.11**	**1.75**	**22.79**	0.43	2.15	162.53
res_2000	108	1404	0.95	5.00	3.42	**0.17**	**1.80**	**21.46**	0.42	2.24	169.70
res_0	108	1410	0.94	5.00	3.43	**0.12**	2.07	**22.11**	0.38	**1.98**	154.44
res_3000	108	1487	0.97	5.00	3.48	**0.20**	**1.99**	**25.29**	0.41	2.38	209.24
res_1000	108	1506	0.94	5.00	3.49	**0.16**	**1.71**	**25.19**	0.40	2.23	213.81
res_2kxa	177	2627	0.93	5.00	3.52	**0.18**	**2.16**	**74.32**	0.39	2.75	621.78
2kxa	177	2711	0.93	5.00	3.53	**0.26**	**2.79**	**78.66**	0.44	3.07	834.17
C0030pkl	198	3247	0.94	5.00	3.56	**0.22**	**2.86**	**115.21**	0.45	3.43	1249.54
cass...-130731	281	4871	0.94	5.00	3.50	**0.21**	**3.03**	**256.58**	0.46	3.46	2029.28
helix_amber	392	6265	0.96	5.00	3.52	**0.26**	**3.30**	**388.29**	0.46	3.63	2600.84
avg						**0.14**	**1.61**	**59.76**	0.31	1.96	459.49
geo (CPU)								**25.43**			156.54

Table 2: Comparative results (MWU vs. MS) on the HARD instances of the DGP. The last line reports the average edge errors and the arithmetic and geometric mean of the CPU times.

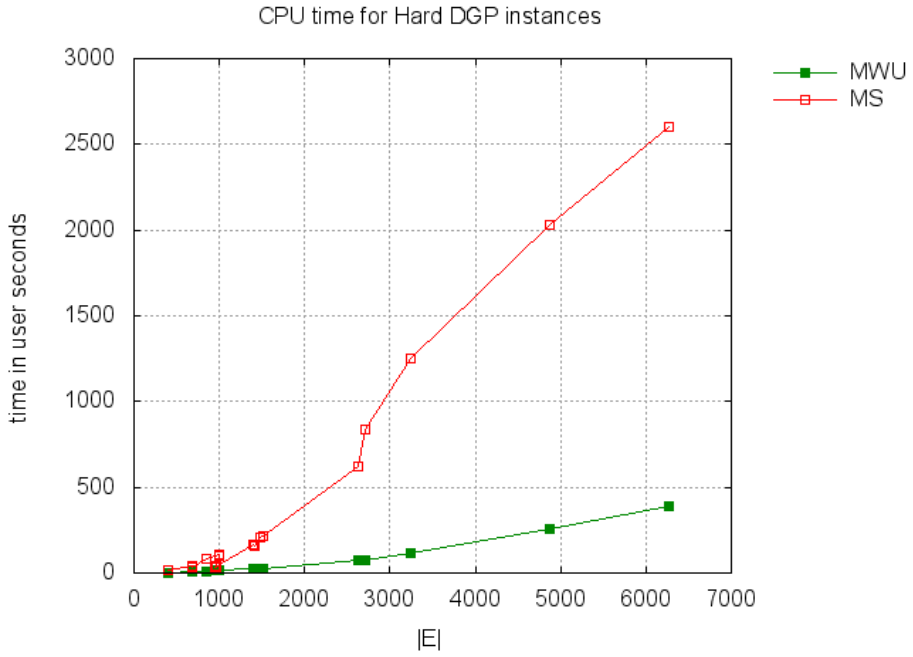


Figure 1: CPU time vs. m: MWU vs. MS.

both quality and CPU time. What is surprising, however, is that the MWU is often able to find a better maximum distance error. This measure is important in the context of proteins, since a single large maximum distance error might well mean a different conformation altogether (possibly closer to a different isomer), and therefore a protein with a different function. On the other hand, the average distance error is somewhat less important (as long as it remains low enough), as it might simply be due to experimental errors. Unlike the MWU, and similarly to MS, VNS has no approximation guarantee whatsoever, even a relative one.

						MWU			VNS		
Instance	n	m	d:min	max	avg	err:avg	max	CPU	avg	max	CPU
2erl-frag-bp1	39	406	0.81	4.99	3.11	0.00	0.00	**2.63**	0.00	0.00	4.00
C0080create.1	60	681	0.98	5.00	3.33	0.04	0.41	**6.17**	**0.00**	**0.00**	9.19
C0080create.2	60	681	0.98	5.00	3.33	0.04	0.41	6.05	**0.00**	**0.00**	**0.7**
names	82	840	1.21	5.00	3.48	0.16	1.28	**10.91**	**0.01**	**0.63**	16.91
pept	107	999	1.16	5.00	3.55	0.16	**1.38**	15.53	**0.15**	1.91	**11.62**
C0020pdb	107	999	1.16	5.00	3.55	0.16	**1.45**	14.69	**0.12**	2.78	**6.44**
1guu-1	150	959	1.20	4.99	3.46	0.09	**0.79**	15.09	**0.04**	1.11	**1.46**
1guu-4000	150	968	0.33	4.99	3.51	**0.11**	19.31	**0.68**	**0.11**	1.08	**10.57**
1guu	150	955	1.30	5.00	3.46	0.09	**0.70**	15.33	**0.06**	1.00	**9.64**
res_5000	108	1392	0.94	5.00	3.42	0.11	**1.75**	22.79	**0.06**	2.57	**10.24**
res_2000	108	1404	0.95	5.00	3.42	0.17	**1.80**	21.46	**0.15**	1.87	**2.50**
res_0	108	1410	0.94	5.00	3.43	**0.12**	**2.07**	22.11	0.2	2.60	**2.32**
res_3000	108	1487	0.97	5.00	3.48	0.20	1.99	25.29	**0.01**	**0.86**	**20.11**
res_1000	108	1506	0.94	5.00	3.49	0.16	**1.71**	25.19	**0.14**	2.89	**10.49**
res_2kxa	177	2627	0.93	5.00	3.52	0.18	2.16	74.32	**0.00**	**0.00**	**20.11**
2kxa	177	2711	0.93	5.00	3.53	0.26	2.79	78.66	**0.01**	**1.26**	**20.86**
C0030pkl	198	3247	0.94	5.00	3.56	0.22	2.86	115.21	**0.00**	**0.01**	**7.80**
cass...-130731	281	4871	0.94	5.00	3.50	0.21	**3.03**	256.58	**0.12**	3.54	**21.09**
helix_amber	392	6265	0.96	5.00	3.52	0.26	**3.30**	388.29	**0.11**	4.52	**23.12**
avg						0.14	1.61	59.76	**0.07**	**1.51**	**11.02**
geo (CPU)								25.43			**7.85**

Table 3: Comparative results (MWU vs. VNS) on the HARD instances of the DGP. The last line reports the average edge errors and the arithmetic and geometric mean of the CPU times.

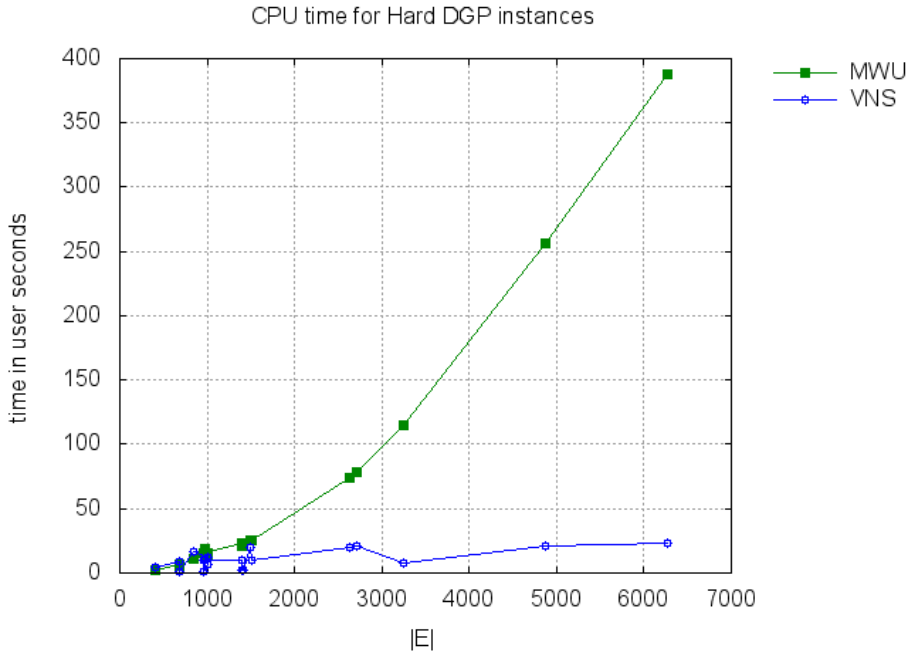


Figure 2: CPU time vs. m: MWU vs. VNS.

4.2 HUC results

In this section we present the results on the HUC application.

4.2.1 Test configuration

We again use the IPOpt solver [14] as the local NLP solver in both the MWU (Step 5 of Alg. 4) and MS (Step 3 of Alg. 1) methods; and Cplex [24] as the LP solver for the pointwise reformulation of the MWU (Step 4 of Alg. 4). External solvers are invoked without time limits. Tests were executed on a machine configured with eight 64-bit Intel Xeon CPU E5504 running at 2.00 GHz and and 11.7 GB of RAM, running the Linux operating system.

The authors of [11] provided us with three simplified instances, sharing the following common characteristics:

- $\bar{h} = 128, \tau = 1\text{h}$
- $V_0 = V_{\bar{h}} = 21078580, \underline{V} = 15000000, \bar{V} = 33000000\text{m}^3$
- $\bar{Q} = 42\text{m}^3/\text{s}$
- $Q^- = Q^+ = 70\text{m}^3/\text{s/hour}$
- $R_0 = 0.01$
- $\underline{L} = 385$
- $L = (L_\ell \mid \ell \in \{0, \dots, 6\}) =$
  $= (4.0986, -1.2554, 0.1605, -9.762 \times 10^{-3}, 3.0943 \times 10^{-4}, -4.9293 \times 10^{-6}, 3.1152 \times 10^{-8})$
- $K = (K_k \mid k \in \{0, \dots, 6\}) =$
  $= (3.074 \times 10^2, 3.88 \times 10^1, -4.37, 0.265, -8.87 \times 10^{-3}, 1.55 \times 10^{-4}, -1.11 \times 10^{-6})$

All of the the values of the inflows I (m^3/s) and prices Π (currency/MWh) per time period are reported in Tab. 8 in the appendix.

In order to work with a larger test set, nine further instances were generated from each of the three original instances by uniformly sampling price vectors $\Pi = (\Pi_h \mid h \in \{1, \dots, 168\})$ in $[\min \Pi, \max \Pi]$. The generated instances are noted `A2` to `A10`, `B2` to `B10`, `C2` to `C10`. For example, instance `C2` features the same data as `C1` except for the prices, which, however, lay in the same range.

4.2.2 Comparative results on solution quality and CPU time

Both MWU and MS are configured with $T = 20$ iterations. Tests are run on the (`A1-C10`) 30-instance set introduced in Sect. 4.2.1. The comparative computational results are reported in Tab. 4 as follows:

- the first column shows the instance name
- second and third columns show objective value and CPU time (in seconds of user time) for the MS algorithm
- the fourth and fifth columns show objective value and CPU time (in seconds of user time) for the MWU algorithm
- the fifth column shows relative objective value improvement from MS to MWU computed as

$$\Delta = \frac{\mathsf{val}(\mathsf{MWU}) - \mathsf{val}(\mathsf{MS})}{\mathsf{val}(\mathsf{MS})} \tag{4.1}$$

- the sixth column shows time improvement ratio from MS to MWU computed as

$$\Lambda = \frac{\mathsf{cpu}(\mathsf{MS})}{\mathsf{cpu}(\mathsf{MWU})}. \tag{4.2}$$

For the last two columns, the comparison metrics are summarized in the last line with the average (avg) and the standard deviation (std) across all 30 instances.

	MS		MWU		MS vs. MWU	
Instance	objective	CPU	objective	CPU	Δ	Λ
A1	4.12E+4	30.2	**4.17E+4**	**5.7**	1.27%	5.35
A2	5.18E+4	33.4	**5.32E+4**	**5.7**	2.59%	5.89
A3	4.88E+4	33.4	**4.97E+4**	**5.6**	1.80%	5.99
A4	4.86E+4	31.8	**4.98E+4**	**5.9**	2.56%	5.42
A5	5.01E+4	34.4	**5.19E+4**	**5.5**	3.63%	6.29
A6	5.00E+4	29.8	**5.10E+4**	**6.1**	2.11%	4.91
A7	5.10E+4	33.0	**5.20E+4**	**5.8**	1.97%	5.68
A8	5.07E+4	33.2	**5.24E+4**	**5.8**	3.37%	5.69
A9	5.08E+4	32.7	**5.21E+4**	**5.7**	2.48%	5.71
A10	5.04E+4	32.0	**5.13E+4**	**6.0**	1.61%	5.32
B1	1.98E+4	29.5	1.98E+4	**4.4**	0.00%	6.70
B2	2.67E+4	33.3	2.67E+4	**5.0**	0.00%	6.64
B3	2.53E+4	30.7	2.53E+4	**4.4**	0.00%	7.03
B4	2.53E+4	29.2	2.53E+4	**4.7**	0.00%	6.22
B5	2.60E+4	30.2	2.60E+4	**5.0**	0.00%	6.05
B6	2.60E+4	28.2	2.60E+4	**4.6**	0.00%	6.11
B7	2.62E+4	28.6	2.62E+4	**4.4**	0.00%	6.54
B8	2.65E+4	29.6	2.65E+4	**4.5**	0.00%	6.54
B9	2.62E+4	29.5	2.62E+4	**4.8**	0.00%	6.20
B10	2.61E+4	31.8	2.61E+4	**4.5**	0.00%	7.08
C1	5.09E+4	26.8	**5.17E+4**	**5.0**	1.60%	5.38
C2	6.82E+4	33.0	**6.99E+4**	**5.7**	2.53%	5.82
C3	6.51E+4	33.6	**6.65E+4**	**5.4**	2.06%	6.29
C4	6.58E+4	31.9	**6.70E+4**	**5.5**	1.85%	5.77
C5	6.70E+4	33.1	**6.87E+4**	**5.0**	2.43%	6.62
C6	6.64E+4	30.2	**6.82E+4**	**5.0**	2.77%	6.07
C7	6.83E+4	31.3	**6.87E+4**	**4.8**	0.55%	6.51
C8	6.83E+4	33.2	**6.86E+4**	**5.0**	0.44%	6.69
C9	6.72E+4	30.9	**6.88E+4**	**4.7**	2.27%	6.62
C10	6.77E+4	32.5	**6.84E+4**	**5.3**	1.03%	6.14
avg	4.67E+4	31.4	**4.75E+4**	**5.2**	**1.36%**	**6.11**
std	1.71E+4	1.9	1.76E+4	0.5	1.19%	0.54
geom	4.33E+4	31.3	**4.39E+4**	**5.1**	NA	**6.09**

Table 4: Objective and CPU time of MS and MWU, relative objective improvement Δ and CPU time improvement Λ from MS to MWU.

The MWU algorithm is always better than the MS algorithm objective-wise but the improvement is relatively small. It must be emphasized, however, that a few percentage points in the objective functions of energy-related optimization problems often translate in consistent savings in absolute terms. As for the time performance, the MWU algorithm is six times faster than the MS algorithm on average, with low variability: this is an extremely desirable feature in short-term unit commitment problems such as this.

4.2.3 Sensitivity to instance size

In order to study in the relative performance sensitivity of the MWU algorithm on instance size, we artificially vary the time horizon $\bar{h}$ of the instances from one day to two weeks.

The instances introduced in Sect. 4.2.1 are one week long with hourly time steps ($\bar{h} = 168$); we now consider instances with $\bar{h}$ in $\{24, 48, \ldots, 168, \ldots, 336\}$, defined as follows:

- when $\bar{h} \leq 168$, data from the 168-long instances is cropped to $h \in [1, \bar{h}]$;
- when $\bar{h} > 168$, data for $h \in [1, 168]$ is identical to the 168-long instances, and data for $h \in [169, \bar{h}]$ is duplicated from the first interval and taken from time period $h - 168 \in [1, 168]$.

Both MWU and MS are configured with $T = 20$. For each instance subset A (A1 to A10), B (B1 to B10) and C (C1 to C10), the comparative computational results for the 14 different sizes are summarized by average and standard deviation in Tab. 5, as follows:

- the first column shows the instance subset
- the second column shows instance size $\bar{h}$

- the third and fourth columns show average (avg) and standard deviation (std) of the CPU time improvement Λ taken over the 10 instances in the first column having size specified in the second
- the fifth and sixth columns show average (avg) and standard deviation (std) of the relative objective improvement Δ taken over the 10 instances in the first column having size specified in the second

For the last three columns, the last line shows average, standard deviation and geometric mean of the corresponding metrics taken over the whole 420 ($= 3 \times 10 \times 14$) tested instances.

Instance subset	Instance size $\bar{h}$	Δ avg	Δ std	Λ avg	Λ std	Λ geo
A	24	**0.38%**	7.73%	**2.76**	0.17	**2.75**
	48	**3.10%**	1.05%	**3.82**	0.17	**3.82**
	72	**3.19%**	1.97%	**4.46**	0.21	**4.46**
	96	**2.30%**	0.88%	**5.12**	0.23	**5.11**
	120	**3.67%**	1.32%	**5.11**	0.23	**5.11**
	144	**1.81%**	1.22%	**5.62**	0.35	**5.61**
	168	**2.23%**	0.66%	**5.73**	0.29	**5.73**
	192	**2.73%**	0.98%	**6.18**	0.33	**6.17**
	216	**2.40%**	1.11%	**6.39**	0.37	**6.38**
	240	**1.86%**	1.25%	**6.48**	0.44	**6.47**
	264	**2.12%**	0.54%	**7.18**	0.40	**7.17**
	288	**2.30%**	0.74%	**6.77**	0.45	**6.76**
	312	**1.84%**	0.73%	**6.65**	0.39	**6.64**
	336	**1.78%**	0.91%	**7.01**	0.50	**7.00**
B	24	0.00%	0.00%	**3.19**	0.24	**3.18**
	48	0.00%	0.00%	**4.87**	0.27	**4.86**
	72	0.00%	0.00%	**5.92**	0.38	**5.91**
	96	0.00%	0.00%	**6.51**	0.38	**6.50**
	120	0.00%	0.00%	**6.31**	0.29	**6.31**
	144	0.00%	0.00%	**6.75**	0.46	**6.74**
	168	0.00%	0.00%	**7.01**	0.46	**6.99**
	192	0.00%	0.00%	**7.41**	0.72	**7.38**
	216	0.00%	0.00%	**7.36**	0.71	**7.33**
	240	0.00%	0.00%	**7.40**	0.31	**7.40**
	264	0.00%	0.00%	**7.68**	0.52	**7.67**
	288	0.00%	0.00%	**7.60**	0.98	**7.54**
	312	0.00%	0.00%	**7.92**	0.75	**7.88**
	336	0.00%	0.00%	**8.45**	0.82	**8.41**
C	24	0.00%	0.00%	**3.06**	0.15	**3.05**
	48	**2.08%**	1.87%	**4.31**	0.21	**4.31**
	72	**2.59%**	2.11%	**4.77**	0.33	**4.76**
	96	**2.99%**	1.01%	**5.17**	0.19	**5.16**
	120	**3.30%**	1.86%	**5.47**	0.23	**5.47**
	144	**2.75%**	0.99%	**5.98**	0.30	**5.97**
	168	**1.89%**	0.72%	**6.59**	0.39	**6.58**
	192	**1.81%**	0.96%	**6.74**	0.60	**6.72**
	216	**1.08%**	0.76%	**6.53**	0.44	**6.52**
	240	**1.70%**	1.34%	**7.41**	0.46	**7.40**
	264	**1.20%**	0.82%	**7.32**	0.85	**7.28**
	288	**1.22%**	0.81%	**7.29**	0.54	**7.27**
	312	**1.46%**	0.85%	**7.31**	0.45	**7.30**
	336	**0.62%**	1.03%	**7.15**	0.46	**7.14**
	avg	**1.34%**	0.86%	**6.16**	0.41	**6.15**
	std	1.20%	1.25%	1.38	0.20	1.37
	geo	NA	NA	**5.98**	0.37	**5.97**

Table 5: Relative objective improvement Δ and CPU time improvement Λ from MS to MWU, according to instance subset and instance size $\bar{h}$.

The results from Tab. 5 corroborate those from Tab. 4: the MWU algorithm almost always outputs a slightly better objective than the MS algorithm while consistently outperforming the MS algorithm in CPU time.

We graphically compare CPU times averaged over instance sizes in Fig. 3. Each point corresponds to (size, average CPU time) over all the instances `A1-C10`.

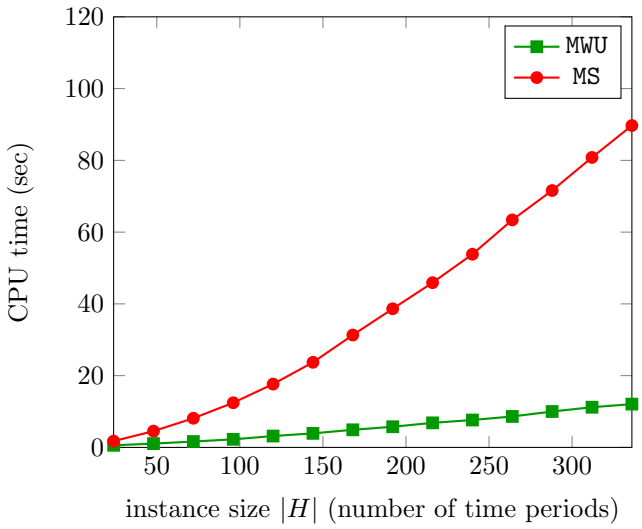


Figure 3: Average of CPU time (sec) vs. instance size $|H| = \bar{h}$. Each point represents a CPU time average over three equally-sized instances (in groups `A`, `B`, `C`).

4.2.4 Influence of initialization

In this section we empirically show that the MWU method is robust to varying inital conditions. We define the *objective dispersion* Ξ as the standard deviation over the average of a sample of objective function values collected from a sequence of runs with randomly chosen starting vectors θ.

The MWU search is somewhat diversified due to the random sampling of θ (Step 3 of Alg. 4), which is used to define the pointwise reformulation. Whenever the ψ costs are only defined through feasibility (i.e. ψ is proportional to β, see Sect. 3.3), it is easy to show that the weights ω can only decrease during MWU execution (Step 8 of Alg. 4), which means the sampling domain is increasingly small. In the extreme case where all weights ω are set to zero for all iterations, the search is deterministic and only depends the values initially set for parameters θ. We therefore test whether MWU results are conditioned by initialization.

To this end, we look at the dispersion of 20 MWU runs (all of them configured with $T = 20$), started from with 20 different randomly sampled vectors θ. In addition, for each sampled initial point, a local descent is independently performed to solve problem Eq. (2.19); the best of these descents is equivalent to the result of a 20-iteration MS run.

Tests are run on the (`A1-C10`) 30-instance set introduced in Sect. 4.2.1 (with $\bar{h} = 168$ time periods). Computational results are reported in Tab. 6, as follows:

- the first column shows the instance name
- the second column shows the objective dispersion Ξ
- the third column shows average objective function value improvements from MS to each of the 20 MWU runs (Δ_{avg})
- the fourth column indicates whether the worst objective function value obtained over the 20 MWU runs is as good as the MS result.

The comparison metrics are summarized in the last line with the average across all 30 instances.

On average, the dispersion of the MWU results is low ($\mathsf{avg}(\Xi) = 0.38\%$), which shows low sensitivity to variability in initialization. In addition, the dispersion of the MWU results is lower on average than the average relative improvement ($\mathsf{avg}(\Delta_{\mathsf{avg}}) = 1.17\%$), thus showing that the MWU is consistently better

Instance	Ξ	Δ_{avg}	MWU $\geq$ MS
A1	0.40%	0.08%	false
A2	0.63%	2.50%	**true**
A3	0.56%	2.76%	**true**
A4	0.83%	1.87%	**true**
A5	0.67%	2.02%	**true**
A6	0.33%	0.20%	false
A7	0.69%	3.93%	**true**
A8	0.57%	1.53%	**true**
A9	0.41%	2.17%	**true**
A10	0.61%	2.02%	**true**
B1	0.00%	0.00%	**true**
B2	0.00%	0.00%	**true**
B3	0.00%	0.00%	**true**
B4	0.00%	0.00%	**true**
B5	0.00%	0.00%	**true**
B6	0.00%	0.00%	**true**
B7	0.00%	0.00%	**true**
B8	0.00%	0.00%	**true**
B9	0.00%	0.00%	**true**
B10	0.00%	0.00%	**true**
C1	0.41%	1.43%	**true**
C2	0.63%	3.44%	**true**
C3	0.45%	1.70%	**true**
C4	0.70%	3.96%	**true**
C5	0.46%	1.40%	**true**
C6	0.52%	0.48%	false
C7	0.57%	2.47%	**true**
C8	0.70%	0.59%	false
C9	0.56%	0.93%	false
C10	0.58%	0.54%	false
avg	**0.38%**	**1.20%**	**80%**

Table 6: Objective dispersion Ξ, average relative objective value improvement from MS Δ_{avg}, and worst-case objective value comparison to MS (last column) for the 20 MWU runs.

than MS; this comparison is all the more relevant as the 20 MWU runs and the 20 MS iterations are started with the same values. Similarly, four out of five times (80%) even the worst MWU run yields a result as good as the MS.

Therefore, the MWU method does not suffer from a mitigated diversification compared to the MS method, and is robust to varying initial conditions.

4.2.5 Importance of the pointwise step

In this section we show that solving the pointwise problem within the MWU method (Step 4 of Alg. 4) contributes to its effectiveness compared to the MS.

Both MWU and MS feature a randomly started local descent. On the one hand, the intialization phase for MS is based on sampling from a uniform distribution. On the other hand, the MWU has a specific routine to sample parameters, which are then used to solve the pointwise problem Eq. (2.20), the solution of which is employed as a starting point to a local descent (refinement step) for the original problem Eq. (2.19). In the MWU loop applied to the HUC, a solution of the pointwise problem is feasible for the original problem (except for the dependent variables y), whereas the initial point randomly sampled in MS may be infeasible. To test if the latter characteristic is responsible for the very good relative time performances of the MWU, we compare results against an enhanced MS with a pointwise feasibility recovery step, as described in Alg. 5.

Both MWU and MSptw are configured with $T = 20$. Tests are run on the (A1-C10) 30-instance set introduced in Sect. 4.2.1, (with $\bar{h} = 168$ time periods). The comparative computational results are reported in Tab. 7 as follows:

- the first column shows instance name
- the second and third columns show objective value and CPU time (in seconds) for the MSptw

Algorithm 5 MSptw(P)

1: **while** $t \leq T$ **do**
2: sample θ^t uniformly at random
3: solve $\underset{t \leftarrow t'(\theta^t)}{\text{ptw}}$ (P), get solution x^t
4: refine x^t (e.g. using local descent)
5: if x^t is better than the incumbent, replace $x^* \leftarrow x^t$
6: increase t
7: **end while**

algorithm

- the fourth and fifth columns show objective value and CPU time (in seconds) for the MWU algorithm
- the fifth column shows relative objective value improvement Δ from MSptw to MWU
- the sixth column shows time improvement ratio Λ from MSptw to MWU.

For the last two columns, the comparison metrics are summarized in the last line with the average (avg) and the standard deviation (std) across all 30 instances.

	MWU		MSptw		MWU vs.	MSptw
Instance	objective	CPU	objective	CPU	Δ	Λ
A1	4.17E+4	**5.48**	**4.18E+4**	7.93	-0.27%	1.45
A2	**5.32E+4**	**5.22**	5.19E+4	8.02	2.40%	1.54
A3	**4.97E+4**	**5.29**	4.95E+4	8.51	0.49%	1.61
A4	**4.98E+4**	**5.51**	4.90E+4	8.3	1.74%	1.51
A5	**5.19E+4**	**5.35**	5.06E+4	8.86	2.56%	1.66
A6	**5.10E+4**	**5.49**	5.06E+4	8.36	0.81%	1.52
A7	**5.20E+4**	**5.3**	5.11E+4	8.3	1.84%	1.57
A8	**5.24E+4**	**5.38**	5.17E+4	7.83	1.36%	1.46
A9	**5.21E+4**	**5.33**	5.14E+4	8.18	1.28%	1.53
A10	**5.13E+4**	**5.57**	5.04E+4	8.54	1.64%	1.53
B1	1.98E+4	**4.2**	1.98E+4	7.33	0.00%	1.75
B2	2.67E+4	**4.57**	2.67E+4	7.52	0.00%	1.65
B3	2.53E+4	**4.11**	2.53E+4	7.64	0.00%	1.86
B4	2.53E+4	**4.37**	2.53E+4	7.21	0.00%	1.65
B5	2.60E+4	**4.8**	2.60E+4	8.28	0.00%	1.73
B6	2.60E+4	**4.35**	2.60E+4	7.8	0.00%	1.79
B7	2.62E+4	**4.1**	2.62E+4	7.8	0.00%	1.90
B8	2.65E+4	**4.28**	2.65E+4	7.97	0.00%	1.86
B9	2.62E+4	**4.65**	2.62E+4	8.04	0.00%	1.73
B10	2.61E+4	**4.33**	2.61E+4	7.25	0.00%	1.67
C1	**5.17E+4**	4.57	5.16E+4	6.7	0.09%	1.47
C2	**6.99E+4**	**5.3**	6.84E+4	8.26	2.11%	1.56
C3	**6.65E+4**	**5.26**	6.57E+4	9.05	1.12%	1.72
C4	**6.70E+4**	**5.64**	6.54E+4	7.97	2.51%	1.41
C5	6.87E+4	**5**	**6.88E+4**	7.9	-0.11%	1.58
C6	**6.82E+4**	**4.9**	6.71E+4	8.25	1.77%	1.68
C7	**6.87E+4**	**5.08**	6.67E+4	8.54	3.02%	1.68
C8	**6.86E+4**	**5.03**	6.79E+4	8.02	1.06%	1.59
C9	**6.88E+4**	**4.7**	6.73E+4	8.18	2.25%	1.74
C10	**6.84E+4**	**5.39**	6.78E+4	8.82	0.96%	1.64
avg	**4.75E+4**	**4.95**	4.70E+4	8.05	**0.95%**	**1.63**
std	1.76E+4	0.49	1.72E+4	0.52	1.02%	0.13
geo	**4.39E+4**	**4.93**	4.35E+4	**8.03**	NA	**1.63**

Table 7: Objective and CPU time of MWU and MS with ptw initialization, relative objective improvement Δ and CPU time improvement Λ from MSptw to MWU.

The MWU method still outperforms the enhanced MSptw method both as regards the objective function value and the CPU time. However, the improvement margin is less marked as with plain MS (see tab 4). Solving a pointwise reformulation to quickly obtain a feasible solution for local descent serves the effectiveness of the MWU method (this is all the more so if the pointwise reformulation is efficient).

4.3 MVPS results

The test set consists in the 20 real-world instances described in [13], publicly available from OR-Library [4, 5] on the web site `http://www.brunel.ac.uk/~mastjjb/jeb/info.html`, differing for the number of assets and for value for the risk level σ . We imposed, as in [13], $\underline{x} = 0.01$, $\bar{x} = 1$ and $K = 10$.

4.3.1 Transaction cost functions, ranked by nonlinearity

We consider different transaction cost functions, which represent different categories of "nonlinearity". In particular, we analyze the performance of the MWU algorithm with respect to the following five univariate functions (see Fig. 4):

(a) $C_i(x_i) = -\rho_i \ln(\frac{20-0.06(1+x_i)}{1+x_i}))$ for all $i \leq n$: this function is increasing, concave and "almost linear".

(b) $C_i(x_i) = -\rho_i \ln(\frac{0.2-0.01(0.00001+x_i)}{0.00001+x_i}))$ for all $i \leq n$: this function is concave and replicates the behavior of the transaction cost function described in [26].

(c) $C_i(x_i) = \rho_i(4x_i + 0.12\sin(40x_i))$ for all $i \leq n$: this function has a sinusoidal behavior similar to a step function.

(d) $C_i(x_i) = \rho_i(4x_i + 0.3\sin(40x_i))$ for all $i \leq n$: this function is similar to the one in (c) but with a "stronger nonlinear behavior".

(e) $C_i(x_i) = \rho_i(0.5x_i + \sin(50x_i))$ for all $i \leq n$: this is the "most nonlinear" transaction cost function among which we tested the methods.

We were careful to use quotes around this informally held quantitative view of the nonlinearity of a function. Since these functions are univariate, we trust most readers will agree with our categorization, by inspection of Fig. 4.

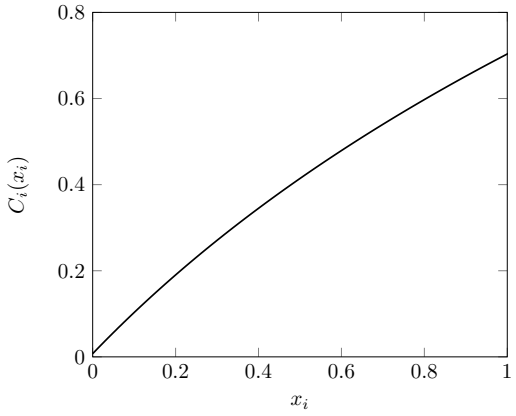
4.3.2 Configuration of the computational platform

We use $T = 20$ iterations of both MWU and MS. We use Bonmin [9] as the local MINLP solver for both methods, with time limit equal to 600 seconds. Specifically, we employ Bonmin's native Branch-and-Bound (B-BB) algorithm [22, 10], since it is generally more stable for nonconvex MINLPs. We use Cplex [24] as the convex MIQP solver for the pointwise reformulation Eq. (2.24), with a 600s time limit, using only one thread. All of the computational experiments were performed on an Intel Xeon CPU E5649, 2.53GHz, using only one processor.

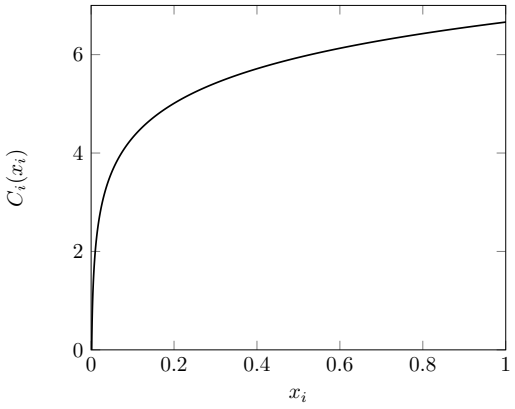
4.3.3 Localization of the MS subsolver

Since the MVPS problem is the only application involving integer variables, a further discussion is in order.

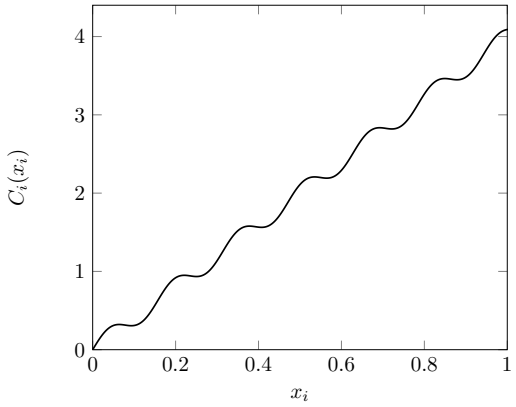
The MS algorithm is based on randomly sampling a starting point, and performing a local optimization starting from that point. On the other hand, since B-BB is a local optimization procedure when deployed on nonconvex MINLP, the only influence of the starting point is that it sets a cut-off value for the optimum. From some preliminary tests, Bonmin behaves more like a global solver on our test-set than a local one, and essentially defies the purpose of comparing the MWU against the MS.



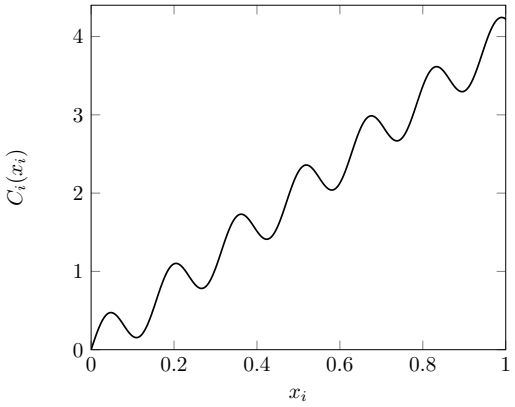
(a) Easy concave transaction costs.



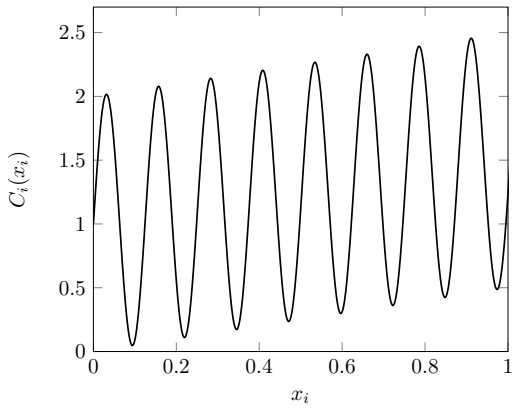
(b) Hard concave transaction costs.



(c) Easy trigonometric transaction costs.



(d) Medium trigonometric transaction costs.



(e) Hard trigonometric transaction costs.

Figure 4: Examples of transaction cost functions.

In order to turn the B-BB into a truly local solver, we added a local branching constraint [20] to the formulation, which limits the amount of flips of the binary variables to a fixed constant $\lfloor \nu n \rfloor$, where $\nu \in [0, 1]$:

$$\sum_{\substack{i \leq n \\ y'_i = 0}} y_i + \sum_{\substack{i \leq n \\ y'_i = 1}} (1 - y_i) \leq \lfloor \nu \, n \rfloor, \tag{4.3}$$

where y' is the starting point. This enforces a local exploration in combinatorial neighbourhood of the

starting point y'. Some trial and error yielded a very high threshold $\nu = 0.96$ (lower values made the instance infeasible excessively often).

4.3.4 Summary of results

The results (see Fig. 5) seem to confirm a preliminary observation that MWU finds better solutions than MS for problems involving "very nonlinear" functions. With transaction costs (a)-(b), MS performs better than MWU; with respect to the transaction costs (c) and (e), however, MWU performs better than MS. In particular, for transaction costs (c)-(d) the average value of relative objective value improvement from MS to MWU is considerably high.

Another interesting (and unexpected) observation concerning solution quality is that, on average, the number of assets in the best portfolio produced by the MWU (a secondary goal when solving such portfolio problems in real life) is smaller with regards to the MS. At the moment we are unable to explain why, so we leave this as an open issue.

As for the geometric mean for the CPU time, MS shows an advantage only in case (a), all the other cases being dominated by MWU.

5 Conclusion

This paper is about the adaptation of the multiplicative weights algorithm to the (nonconvex) nonlinear and mixed integer nonlinear programming setting, based on a particular parametrized reformulation of the problem. Though this algorithm has been previously employed as a theoretical tool to derive approximation algorithms, we decided to benchmark it computationally to solve three classes of hard problems from different application fields. We found it compares quite favorably to the well-known multi-start method, which, unlike the MWU, offers no approximation guarantee.

Acknowledgments

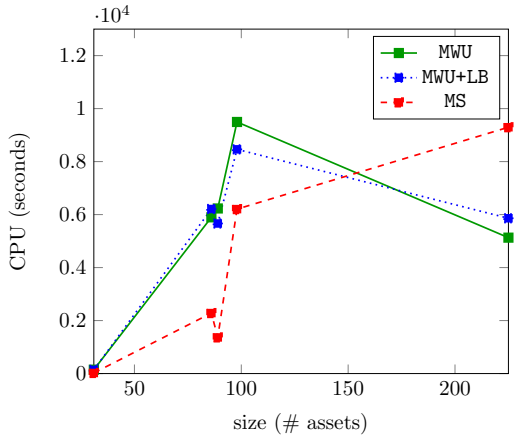
We are very grateful to Dr. Pascale Bendotti (EDF) for useful suggestions about the HUC problem. Luca Mencarelli is sponsored by a Ph.D. Fellowship from the FP7 Marie Curie ITN "MINO" project. Youcef Sahraoui is sponsored by a CIFRE Ph.D. Fellowship with Éléctricité De France (EDF). Leo Liberti was partly sponsored by the ANR Bip:Bip project under contract ANR-10-BINF-0003, and completed this work during a visiting term at IMECC, University of Campinas (SP), Brazil, sponsored by the Chaires Françaises dans l'état de São Paulo (CFSP) program, a collaboration between the French Consulate in São Paulo and the three main universities in São Paulo: UNICAMP, USP and UNESP.

References

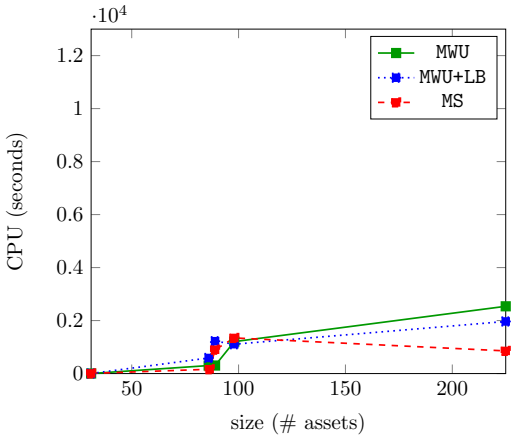
[1] Arora, S., Hazan, E., Kale, S.: Fast algorithms for approximate semidefinite programming using the multiplicative weights update method. In: Foundations of Computer Science, *FOCS*, vol. 46, pp. 339–348. IEEE (2005)

[2] Arora, S., Hazan, E., Kale, S.: The multiplicative weights update method: a meta-algorithm and applications. Theory of Computing **8**, 121–164 (2012)

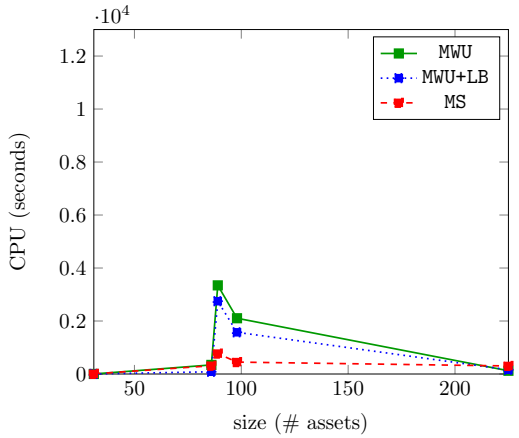
[3] Bahr, A., Leonard, J., Fallon, M.: Cooperative localization for autonomous underwater vehicles. International Journal of Robotics Research **28**(6), 714–728 (2009)



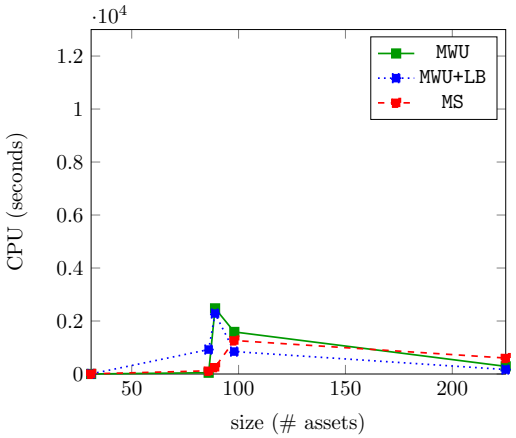
(a) "Easy" concave transaction costs.



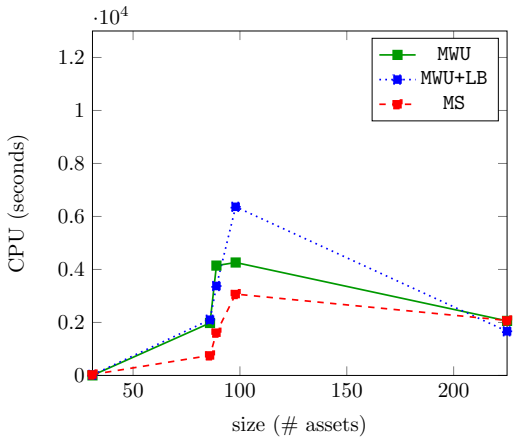
(b) "Hard" concave transaction costs.



(c) "Easy" trigonometric transaction costs.



(d) "Medium" trigonometric transaction costs.



(e) "Hard" trigonometric transaction costs.

Figure 5: CPU time vs. size of the problem n (# assets).

[4] Beasley, J.: OR-Library: Distributing test problems by electronic mail. Journal of the Operational Research Society **41**(11), 1069–1072 (1990)

[5] Beasley, J.: Obtaining test problems via internet. Journal of Global Optimization **8**(4), 429–433

(1996)

[6] Beeker, N., Gaubert, S., Glusa, C., Liberti, L.: Is the distance geometry problem in **NP**? In: A. Mucherino, C. Lavor, L. Liberti, N. Maculan (eds.) Distance Geometry: Theory, Methods, and Applications. Springer, New York (2013)

[7] Berman, H., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T., Weissig, H., Shindyalov, I., Bourne, P.: The protein data bank. Nucleic Acid Research **28**, 235–242 (2000)

[8] Bienstock, D.: Computational study of a family of mixed-integer quadratic programming problems. Mathematical Programming **74**(2), 121–140 (1996)

[9] Bonami, P., Lee, J.: `BONMIN` user's manual. Tech. rep., IBM Corporation (2007)

[10] Bonami, P., Lee, J., Leyffer, S., Waecher, A.: More Branch-and-Bound experiments in convex nonlinear integer programming (2011). Preprint ANL/MCS-P1949-0911, Argonne National Laboratory, Mathematics and Computer Science Division

[11] Borghetti, A., D'Ambrosio, C., Lodi, A., Martello, S.: Optimal scheduling of a multiunit hydro power station in a short-term planning horizon. In: K.G. Murty (ed.) Case Studies in Operations Research, *International Series in Operations Research & Management Science*, vol. 212, pp. 167–181. Springer New York (2015). DOI 10.1007/978-1-4939-1007-6_8. URL `http://dx.doi.org/10.1007/978-1-4939-1007-6_8`

[12] Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, Cambridge (2004)

[13] Chang, T.J., Meade, N., Beasley, J., Sharaiha, Y.: Heuristics for cardinality constrained portfolio optimization. Computers and Operations Research **27**(13), 1271–1302 (2000)

[14] COIN-OR: Introduction to IPOPT: A tutorial for downloading, installing, and using IPOPT (2006)

[15] Costa, A., Hansen, P., Liberti, L.: Formulation symmetries in circle packing. In: R. Mahjoub (ed.) Proceedings of the International Symposium on Combinatorial Optimization, *Electronic Notes in Discrete Mathematics*, vol. 36, pp. 1303–1310. Elsevier, Amsterdam (2010)

[16] D'Ambrosio, C., Ky, V., Lavor, C., Liberti, L., Maculan, N.: Solving distance geometry problems with interval data using formulation-based methods. Tech. rep., LIX Ecole Polytechnique (working paper) (2014)

[17] D'Ambrosio, C., Mencarelli, L.: Complex portfolio selection via convex mixed-integer quadratic approaches: A survey. Tech. rep., LIX, École Polytechnique (working paper) (2014)

[18] Ding, Y., Krislock, N., Qian, J., Wolkowicz, H.: Sensor network localization, Euclidean distance matrix completions, and graph realization. Optimization and Engineering **11**, 45–66 (2010)

[19] Du, H., Alechina, N., Stock, K., Jackson, M.: The logic of NEAR and FAR. In: T.T. et al. (ed.) COSIT, *LNCS*, vol. 8116, pp. 475–494. Springer, Switzerland (2013)

[20] Fischetti, M., Lodi, A.: Local branching. Mathematical Programming, Series B **98**(1–3), 23–47 (2003)

[21] Frangioni, A., Gentile, C.: Perspective cuts for a class of convex 0-1 mixed integer programs. Mathematical Programming, Series A **106**(2), 225–236 (2006)

[22] Gupta, O., Ravindran, A.: Branch-and-Bound experiments in convex nonlinear integer programming. Management Science **31**(12), 1533–1546 (1985)

[23] Hansen, P., Mladenović, N.: Variable neighbourhood search: Principles and applications. European Journal of Operations Research **130**, 449–467 (2001)

[24] IBM: ILOG CPLEX 12.2 User's Manual. IBM (2010)

[25] Kannan, R., Monma, C.: On the computational complexity of integer programming problems. In: R. Henn, B. Korte, W. Oettli (eds.) Optimization and Operations Research, *Lecture Notes in Economics and Mathematical Systems*, vol. 157, pp. 161–172. Springer–Verlag (1978)

[26] Konno, H., Wijayanayake, A.: Portfolio optimization problem under concave transaction costs and minimal transaction unit constraints. Mathematical Programming, Series B **89**(2), 233–250 (2001)

[27] Lavor, C., Liberti, L., Maculan, N.: Computational experience with the molecular distance geometry problem. In: J. Pintér (ed.) Global Optimization: Scientific and Engineering Case Studies, pp. 213–225. Springer, Berlin (2006)

[28] Liberti, L.: Reformulations in mathematical programming: Definitions and systematics. RAIRO-RO **43**(1), 55–86 (2009)

[29] Liberti, L., Lavor, C., Maculan, N., Mucherino, A.: Euclidean distance geometry and applications. SIAM Review **56**(1), 3–69 (2014)

[30] Malliavin, T., Mucherino, A., Nilges, M.: Distance geometry in structural biology. In: A. Mucherino, C. Lavor, L. Liberti, N. Maculan (eds.) Distance Geometry: Theory, Methods, and Applications. Springer, New York (2013)

[31] Maniezzo, V., Stützle, T., Voß, S. (eds.): Hybridizing metaheuristics and mathematical programming, *Annals of Information Systems*, vol. 10. Springer, New York (2009)

[32] Markowitz, H.: Portfolio selection. The Journal of Finance **7**(1), 77–91 (1952)

[33] Plotkin, S., Shmoys, D., Tardos, E.: Fast approximation algorithm for fractional packing and covering problems. Mathematics of Operations Research **20**, 257–301 (1995)

[34] Saxe, J.: Embeddability of weighted graphs in k-space is strongly **NP**-hard. Proceedings of 17th Allerton Conference in Communications, Control and Computing pp. 480–489 (1979)

[35] Scherer, B., Martin, D.: Introduction to Modern Portfolio Optimization. Springer–Verlag (2005)

[36] Schlick, T.: Molecular modelling and simulation: an interdisciplinary guide. Springer, New York (2002)

[37] Shaw, D., Liu, S., Kopman, L.: Lagrangian relaxation procedure for cardinality-constrained portfolio optimization. Optimization Methods and Software **23**(3), 411–420 (2008)

[38] Singer, A.: Angular synchronization by eigenvectors and semidefinite programming. Applied and Computational Harmonic Analysis **30**, 20–36 (2011)

[39] Sun, X., Zheng, X., Li, D.: Recent advances in mathematical programming with semi-continuous variables and cardinality constraint. Journal of the Operations Research Society of China **1**(1), 55–77 (2013)

[40] Tahanan, M., van Ackooij, W., Frangioni, A., Lacalandra, F.: Large-scale unit commitment under uncertainty: a literature survey. 4OR **13**, 115–171 (2015)

[41] Xue, H.G., Xu, G.X., Feng, Z.X.: Mean-variance portfolio optimal problem under concave transaction cost. Applied Mathematics and Computation **174**(1), 1–12 (2006)

A Parameter values for the HUC instances

	instance A1		instance B1		instance C1	
h	I_h	Π_h	I_h	Π_h	I_h	Π_h
1	2.66	50.17	0.53	42.37	1.53	48.06
2	2.66	40.17	0.53	29.75	1.53	48.06
3	2.66	35.17	0.53	30.09	1.53	47.56
4	2.66	35.17	0.53	30.12	1.53	47.00

	instance A1		instance B1		instance C1	
h	I_h	Π_h	I_h	Π_h	I_h	Π_h
85	2.19	57.99	0.66	84.77	3.06	71.69
86	2.19	61.75	0.66	92.72	3.06	57.64
87	2.19	61.69	0.66	99.75	3.06	55.62
88	2.19	56.24	0.66	103.72	3.06	55.62

	instance A1		instance B1		instance C1	
h	I_h	Π_h	I_h	Π_h	I_h	Π_h
5	2.66	35.15	0.53	30.20	1.53	47.55
6	2.66	40.05	0.53	30.29	1.53	47.73
7	2.66	57.17	0.53	52.40	1.53	54.20
8	2.66	75.17	0.53	60.24	1.53	75.00
9	2.66	90.00	0.53	93.60	1.53	105.00
10	2.66	147.00	0.53	140.11	1.53	110.61
11	2.66	146.94	0.53	148.11	1.53	110.63
12	2.66	139.95	0.53	141.11	1.53	100.61
13	2.66	95.00	0.53	82.61	1.53	78.61
14	2.66	90.19	0.53	77.61	1.53	77.71
15	2.66	95.00	0.53	90.61	1.53	94.91
16	2.66	95.36	0.53	103.61	1.53	95.63
17	2.66	90.61	0.53	107.60	1.53	188.11
18	2.66	75.49	0.53	97.60	1.53	199.13
19	2.66	60.39	0.53	73.61	1.53	199.13
20	2.66	80.50	0.53	62.61	1.53	110.63
21	2.66	95.62	0.53	61.61	1.53	79.63
22	2.66	65.25	0.53	75.45	1.53	61.62
23	2.51	60.25	0.52	61.25	1.53	58.49
24	2.36	55.05	0.51	59.15	1.52	52.85
25	2.21	50.24	0.50	41.75	1.52	48.60
26	2.06	40.16	0.49	30.24	1.52	48.49
27	1.91	35.15	0.48	30.14	1.52	48.49
28	1.91	35.07	0.48	30.12	1.52	48.38
29	1.91	35.09	0.48	29.75	1.52	48.38
30	1.91	40.16	0.48	29.75	1.52	48.62
31	1.91	57.06	0.48	52.40	1.52	54.62
32	1.91	65.23	0.48	60.13	1.52	75.62
33	1.91	100.61	0.48	93.00	1.52	105.63
34	1.91	147.61	0.48	132.00	1.52	110.63
35	1.91	147.62	0.48	140.00	1.52	105.63
36	1.91	130.61	0.48	133.00	1.52	100.62
37	1.91	95.61	0.48	81.93	1.52	75.63
38	1.91	80.59	0.48	77.14	1.52	75.63
39	1.91	95.61	0.48	90.22	1.52	95.62
40	1.91	95.60	0.48	110.00	1.52	95.63
41	1.91	90.60	0.48	110.00	1.52	185.15
42	1.91	85.60	0.48	96.77	1.52	199.27
43	1.91	70.60	0.48	72.75	1.52	199.27
44	1.91	70.61	0.48	61.75	1.52	110.64
45	1.91	95.61	0.48	74.75	1.52	75.65
46	1.91	80.59	0.48	60.93	1.52	75.63
47	1.82	60.48	0.48	60.88	1.74	70.62
48	1.73	55.29	0.48	58.69	1.96	53.63
49	1.64	55.73	0.48	50.75	2.18	48.63
50	1.56	44.75	0.48	30.60	2.40	48.61
51	1.47	39.75	0.48	30.33	2.63	48.61
52	1.47	34.75	0.48	30.25	2.63	48.61
53	1.47	39.68	0.48	37.25	2.63	48.60
54	1.47	50.59	0.48	40.25	2.63	47.79
55	1.47	52.73	0.48	52.04	2.63	59.55
56	1.47	62.00	0.48	61.10	2.63	75.68
57	1.47	96.00	0.48	88.63	2.63	95.91
58	1.47	145.50	0.48	101.64	2.63	105.97
59	1.47	145.49	0.48	140.60	2.63	105.90
60	1.47	145.00	0.48	121.64	2.63	95.87
61	1.47	94.49	0.48	84.40	2.63	71.82
62	1.47	85.29	0.48	89.40	2.63	59.60
63	1.47	89.34	0.48	90.61	2.63	59.57
64	1.47	85.49	0.48	104.63	2.63	64.50
65	1.47	88.89	0.48	100.64	2.63	74.78
66	1.47	79.70	0.48	94.62	2.63	95.80
67	1.47	61.61	0.48	76.40	2.63	99.89
68	1.47	83.75	0.48	68.40	2.63	95.80
69	1.47	92.75	0.48	64.60	2.63	77.00
70	1.47	80.07	0.48	76.40	2.63	69.64
71	1.61	63.95	0.52	68.40	2.71	59.67
72	1.76	61.24	0.55	56.61	2.80	48.61
73	1.90	62.60	0.59	50.62	2.89	52.62
74	2.05	52.65	0.62	29.61	2.97	50.53
75	2.19	36.65	0.66	29.62	3.06	47.64

	instance A1		instance B1		instance C1	
h	I_h	Π_h	I_h	Π_h	I_h	Π_h
89	2.19	56.24	0.66	99.75	3.06	71.62
90	2.19	51.70	0.66	93.68	3.06	96.89
91	2.19	55.63	0.66	76.65	3.06	96.92
92	2.19	64.64	0.66	63.71	3.06	97.00
93	2.19	70.00	0.66	68.75	3.06	96.97
94	2.19	61.00	0.66	76.75	3.06	74.92
95	2.34	57.12	0.67	68.62	2.84	67.67
96	2.48	56.02	0.67	55.67	2.62	50.62
97	2.63	40.50	0.68	60.61	2.39	48.62
98	2.78	40.50	0.68	50.61	2.17	47.63
99	2.93	35.30	0.69	45.61	1.95	47.54
100	2.93	35.31	0.69	30.61	1.95	47.00
101	2.93	30.25	0.69	30.61	1.95	47.00
102	2.93	30.26	0.69	30.61	1.95	47.43
103	2.93	40.25	0.69	41.61	1.95	49.62
104	2.93	48.25	0.69	56.34	1.95	75.82
105	2.93	48.25	0.69	76.61	1.95	92.85
106	2.93	60.56	0.69	92.50	1.95	109.51
107	2.93	60.71	0.69	92.41	1.95	109.02
108	2.93	60.86	0.69	79.41	1.95	93.93
109	2.93	60.50	0.69	61.49	1.95	76.00
110	2.93	47.45	0.69	58.17	1.95	75.92
111	2.93	47.40	0.69	58.00	1.95	76.45
112	2.93	47.45	0.69	58.75	1.95	99.92
113	2.93	51.45	0.69	59.75	1.95	184.03
114	2.93	60.40	0.69	59.75	1.95	199.14
115	2.93	60.40	0.69	59.75	1.95	199.14
116	2.93	60.45	0.69	59.75	1.95	120.44
117	2.93	79.71	0.69	70.11	1.95	80.47
118	2.93	69.89	0.69	70.11	1.95	72.43
119	2.63	60.45	0.71	56.75	1.92	59.64
120	2.34	57.45	0.73	50.00	1.88	49.70
121	2.05	40.25	0.75	55.75	1.85	47.70
122	1.76	40.11	0.77	30.48	1.82	47.55
123	1.47	35.22	0.80	30.50	1.79	47.00
124	1.47	30.25	0.80	30.49	1.79	47.00
125	1.47	30.25	0.80	30.50	1.79	47.00
126	1.47	35.29	0.80	30.60	1.79	47.58
127	1.47	40.25	0.80	30.50	1.79	53.69
128	1.47	47.45	0.80	30.50	1.79	75.25
129	1.47	48.26	0.80	50.61	1.79	118.00
130	1.47	60.08	0.80	60.60	1.79	138.10
131	1.47	60.62	0.80	60.50	1.79	118.50
132	1.47	60.78	0.80	60.50	1.79	100.29
133	1.47	60.61	0.80	60.49	1.79	78.25
134	1.47	47.87	0.80	50.61	1.79	78.07
135	1.47	47.45	0.80	50.40	1.79	95.50
136	1.47	48.05	0.80	50.23	1.79	95.63
137	1.47	51.62	0.80	50.20	1.79	184.94
138	1.47	60.08	0.80	50.20	1.79	199.13
139	1.47	60.40	0.80	50.25	1.79	198.50
140	1.47	60.66	0.80	50.40	1.79	100.63
141	1.47	79.86	0.80	60.25	1.79	79.35
142	1.47	70.00	0.80	70.21	1.79	74.89
143	1.58	60.89	0.74	60.00	1.67	69.63
144	1.70	61.00	0.68	59.84	1.56	52.75
145	1.81	40.40	0.62	53.00	1.44	48.00
146	1.93	40.07	0.57	30.25	1.33	47.68
147	2.04	35.07	0.51	30.25	1.22	47.50
148	2.04	29.63	0.51	29.75	1.22	45.51
149	2.04	29.68	0.51	29.40	1.22	46.87
150	2.04	40.07	0.51	29.4	1.22	48.02
151	2.04	56.75	0.51	30.24	1.22	54.61
152	2.04	70.16	0.51	59.40	1.22	75.49
153	2.04	90.47	0.51	94.75	1.22	118.62
154	2.04	147.43	0.51	110.00	1.22	118.62
155	2.04	147.28	0.51	110.06	1.22	118.62
156	2.04	140.21	0.51	100.45	1.22	100.62
157	2.04	75.20	0.51	87.77	1.22	78.02
158	2.04	80.00	0.51	87.88	1.22	78.62
159	2.04	90.28	0.51	92.00	1.22	95.62

	instance A1		instance B1		instance C1	
h	I_h	Π_h	I_h	Π_h	I_h	Π_h
76	2.19	33.63	0.66	29.62	3.06	47.62
77	2.19	30.64	0.66	36.51	3.06	47.56
78	2.19	32.61	0.66	39.62	3.06	47.56
79	2.19	52.68	0.66	51.65	3.06	47.67
80	2.19	57.75	0.66	64.61	3.06	50.09
81	2.19	59.39	0.66	87.68	3.06	49.44
82	2.19	69.99	0.66	100.81	3.06	71.66
83	2.19	70.00	0.66	139.82	3.06	74.00
84	2.19	65.00	0.66	109.88	3.06	72.59

	instance A1		instance B1		instance C1	
h	I_h	Π_h	I_h	Π_h	I_h	Π_h
160	2.04	95.27	0.51	92.11	1.22	95.62
161	2.04	90.25	0.51	92.18	1.22	185.12
162	2.04	75.10	0.51	82.00	1.22	199.12
163	2.04	74.79	0.51	81.75	1.22	199.12
164	2.04	84.89	0.51	60.75	1.22	100.62
165	2.04	109.89	0.51	74.77	1.22	79.62
166	2.04	70.04	0.51	65.45	1.22	75.62
167	2.04	55.19	0.51	55.07	1.22	70.00
168	2.04	54.67	0.51	51.00	1.22	53.07

Table 8: Inflows I_h (m^3/s) and prices Π_h (currency/MWh) for time periods $h \in \{1, \ldots, \bar{h} = 164\}$ for the 3 provided instances.

B Detailed results for the MVPS problem

Tables 10-13 report the numeric results for each transaction cost function. Their columns are as follows:

- instance name;
- maximum risk level σ;
- number n of assets quoted on the financial market;
- objective value for the MWU algorithm;
- CPU time (in seconds) for the MWU algorithm;
- objective value for the MWU algorithm with the local branching constraint;
- CPU time (in seconds) for the MWU algorithm with the local branching constraint;
- objective value for the MS algorithm with the local branching constraint;
- CPU time (in seconds) for the MS algorithm with the local branching constraint;
- relative objective value improvement from MS to MWU computed as

$$\Gamma = \frac{\mathsf{val(MWU)} - \mathsf{val(MS)}}{|\mathsf{val(MS)}|}; \tag{B.1}$$

- time improvement ratio Λ from MS to MWU (see Eq. (4.2));
- relative objective value improvement from MS to MWU with the local branching constraint (see Eq. (B.1));
- time improvement ratio Λ from MS to MWU with the local branching constraint (see Eq. (4.2)).

The comparison metrics are summarized in the last three lines with the sum ($\sum$), average (avg) and the standard deviation (std) across all 20 instances.

			MWU			MWU+LB			MS			MWU vs. MS		MWU+LB vs. MS	
Instance	σ	n	objective	CPU	$\|x^*\|_0$	objective	CPU	$\|x^*\|_0$	objective	CPU	$\|x^*\|_0$	Γ	Λ	Γ	Λ
port-orlib1_000	0.0047755010	31	**1.0742596**	292.556	**1**	**1.0742596**	314.918	**1**	**1.0742596**	**31.203**	**1**	0.000%	0.107	0.000%	0.099
port-orlib1_050	0.0021522075	31	**1.0742596**	268.278	**1**	**1.0742596**	212.915	**1**	**1.0742596**	**31.469**	**1**	0.000%	0.117	0.000%	0.148
port-orlib1_100	0.0010585969	31	**1.0592082**	13.587	**3**	**1.0592082**	**12.915**	**3**	**1.0592082**	13.180	**3**	0.000%	0.970	0.000%	1.021
port-orlib1_150	0.0007158421	31	**1.0180977**	11.998	**5**	**1.0180977**	12.391	**5**	**1.0180977**	**11.764**	**5**	0.000%	0.980	0.000%	0.949
port-orlib2_000	0.0028352430	85	**1.8750415**	16000.371	**1**	**1.8750415**	18516.395	**1**	**1.8750415**	**1339.651**	**1**	0.000%	0.084	0.000%	0.072
port-orlib2_050	0.0004953237	85	**1.8741070**	330.941	**3**	**1.8741070**	196.865	**3**	**1.8741070**	**182.681**	**3**	0.000%	0.552	0.000%	0.928
port-orlib2_100	0.0002704062	85	**1.8587348**	7038.599	**5**	**1.8587348**	**5745.031**	**5**	**1.8587348**	7183.440	**5**	0.000%	1.021	0.000%	1.250
port-orlib2_150	0.0001663200	85	**1.7828905**	**214.691**	**9**	**1.7828905**	384.026	**9**	**1.7828905**	425.750	**9**	0.000%	1.983	0.000%	1.109
port-orlib3_000	0.0015166351	89	**2.3898574**	18019.824	**1**	**2.3898574**	17862.785	**1**	**2.3898574**	**1396.067**	**1**	0.000%	0.077	0.000%	0.078
port-orlib3_050	0.0005849758	89	2.3892174	2831.204	**2**	**2.3892246**	**213.402**	**2**	2.3892174	333.062	**2**	0.000%	0.118	0.000%	1.561
port-orlib3_100	0.0003215941	89	2.3622650	743.189	8	2.3622650	**173.656**	8	**2.3645800**	490.434	**7**	-0.098%	0.660	-0.098%	2.824
port-orlib3_150	0.0002239117	89	2.2652696	**3329.717**	**10**	2.2645169	4405.874	**10**	**2.2677086**	3238.561	**10**	-0.108%	0.973	-0.141%	0.735
port-orlib4_000	0.0029387241	98	2.9234597	29875.539	**1**	2.9234597	26538.931	**1**	**2.9234600**	**4777.167**	**1**	0.000%	0.160	0.000%	0.180
port-orlib4_050	0.0006828450	98	**2.9200045**	210.979	**3**	**2.9200045**	**192.266**	**3**	**2.9200045**	742.838	**3**	0.000%	3.521	0.000%	3.864
port-orlib4_100	0.0003059553	98	2.9067690	6997.871	5	2.9065884	**6364.984**	**4**	**2.9073338**	11884.665	5	-0.019%	1.698	-0.026%	1.867
port-orlib4_150	0.0001613979	98	**2.8526192**	895.863	10	**2.8526192**	**754.278**	10	**2.8526192**	6461.036	10	0.000%	7.212	0.000%	8.566
port-orlib5_000	0.0016485224	225	**4.6677119**	**3478.872**	**2**	**4.6677119**	5288.969	2	4.6673823	5736.267	4	0.007%	1.649	0.007%	1.085
port-orlib5_050	0.0005150277	225	**4.6668554**	5652.420	**3**	**4.6668554**	**4949.565**	**3**	**4.6668554**	10932.209	**3**	0.000%	1.934	0.000%	2.209
port-orlib5_100	0.0003918260	225	**4.6577793**	**3451.980**	**4**	**4.6577793**	3640.422	**4**	**4.6577793**	9833.002	**4**	0.000%	2.849	0.000%	2.701
port-orlib5_150	0.0003272876	225	4.6264568	**7934.132**	**8**	**4.6228591**	9564.905	**9**	4.6264568	10684.629	**8**	0.000%	1.347	-0.078%	1.117
$\sum$			51.2448641	107592.611	**85**	51.2403402	105345.493	**85**	**51.2498536**	**75729.075**	86	-0.218%	**28.011**	-0.335%	**32.362**
avg			2.5622432	5379.631	**4.250**	2.5620170	5267.275	**4.250**	**2.5624927**	**3786.454**	4.300	-0.011%	**1.401**	-0.017%	**1.618**
std			(1.2414709)	(7721.167)	(63.143)	(1.2411631)	(7478.113)	(3.210)	(1.2413996)	(4287.608)	(3.045)	(0.032%)	(1.672)	(0.040%)	(1.928)
geo (CPU)				1324.256			1090.151			**909.83**					

Table 9: Comparative results of MS and MWU for the transaction cost function (a).

			MWU			MWU+LB			MS			MWU vs. MS		MWU+LB vs. MS	
Instance	σ	n	objective	CPU	$\|x^*\|_0$	objective	CPU	$\|x^*\|_0$	objective	CPU	$\|x^*\|_0$	Γ	Λ	Γ	Λ
port-orlib1_000	0.0047755010	31	**0.3283893**	1.502	**1**	**0.3283893**	**1.295**	**1**	**0.3283893**	4.896	1	0.000%	3.260	0.000%	3.781
port-orlib1_050	0.0021522075	31	0.3268002	1.698	2	0.3268002	**1.488**	2	**0.3268555**	5.310	**1**	-0.017%	3.127	-0.017%	3.569
port-orlib1_100	0.0010585969	31	**0.3264242**	**1.807**	**3**	**0.3264242**	1.977	**3**	**0.3264242**	5.493	**3**	0.000%	3.040	0.000%	2.778
port-orlib1_150	0.0007158421	31	**0.3255758**	**2.890**	**6**	**0.3255758**	3.002	**6**	**0.3255758**	5.649	**6**	0.000%	1.955	0.000%	1.882
port-orlib2_000	0.0028352430	85	**0.5696578**	7.404	**1**	**0.5696578**	**7.059**	**1**	**0.5696578**	43.570	1	0.000%	5.885	0.000%	6.172
port-orlib2_050	0.0004953237	85	**0.5677078**	19.638	**5**	0.5677075	**11.335**	8	0.5677077	52.401	**5**	0.000%	2.668	0.000%	4.623
port-orlib2_100	0.0002704062	85	0.5671808	611.424	**8**	0.5671809	776.712	10	**0.5671812**	**35.439**	10	0.000%	0.058	0.000%	0.046
port-orlib2_150	0.0001663200	85	**0.5668838**	**568.841**	**10**	**0.5668838**	1560.205	**10**	**0.5668838**	1737.240	**10**	0.000%	3.054	0.000%	1.113
port-orlib3_000	0.0015166351	89	**0.7248280**	**10.288**	**1**	**0.7248280**	13.463	**1**	**0.7248280**	37.717	**1**	0.000%	3.666	0.000%	2.802
port-orlib3_050	0.0005849758	89	0.7234229	19.936	3	**0.7234556**	**19.177**	**2**	**0.7234556**	46.714	**2**	-0.005%	2.343	0.000%	2.436
port-orlib3_100	0.0003215941	89	**0.7229391**	**17.926**	**7**	**0.7229391**	1210.131	**7**	0.7229587	651.578	**7**	-0.003%	36.348	-0.003%	0.538
port-orlib3_150	0.0002239117	89	**0.7225929**	**1165.350**	**10**	0.7225928	3643.605	**10**	**0.7225929**	2864.165	**10**	0.000%	2.458	0.000%	0.786
port-orlib4_000	0.0029387241	98	**0.8862800**	**11.850**	**1**	**0.8862800**	29.022	**1**	**0.8862800**	50.988	**1**	0.000%	4.303	0.000%	1.757
port-orlib4_050	0.0006828450	98	**0.8846300**	**28.928**	**2**	**0.8846300**	50.449	**2**	**0.8846300**	61.230	**2**	0.000%	2.117	0.000%	1.214
port-orlib4_100	0.0003059553	98	0.8840668	1005.380	**7**	0.8840668	**44.561**	**7**	**0.8840702**	681.456	9	0.000%	0.678	0.000%	15.293
port-orlib4_150	0.0001613979	98	0.8836949	**3778.496**	**10**	0.8836945	4306.084	**10**	**0.8836971**	4254.159	**10**	0.000%	1.126	0.000%	0.988
port-orlib5_000	0.0016485224	225	**1.4120658**	211.948	**2**	**1.4120658**	**117.720**	**2**	1.4120587	381.806	3	0.001%	1.801	0.001%	3.243
port-orlib5_050	0.0005150277	225	**1.4112207**	4895.417	**5**	**1.4112207**	3055.888	**5**	**1.4112207**	**892.922**	**5**	0.000%	0.182	0.000%	0.292
port-orlib5_100	0.0003918260	225	**1.4109834**	4291.445	**7**	**1.4109834**	2450.898	**7**	**1.4109834**	**777.179**	**7**	0.000%	0.181	0.000%	0.317
port-orlib5_150	0.0003272876	225	**1.4108196**	**755.727**	**7**	1.4108195	2242.568	**7**	**1.4108196**	1368.145	**7**	0.000%	1.810	0.000%	0.610
$\sum$			15.6561636	17407.895	**98**	15.6561956	19546.639	102	**15.6562699**	**13958.057**	101	-0.024%	**80.060**	-0.020%	**54.240**
avg			0.7828082	870.395	**4.900**	0.7828098	977.332	5.100	**0.7828135**	**697.903**	5.050	-0.001%	**4.003**	-0.001%	**2.712**
std			(0.3736084)	(1542.238)	(3.243)	(0.3736081)	(1406.957)	(3.478)	(0.3736038)	(1121.600)	(3.517)	(0.004%)	(7.751)	(0.004%)	3.380
geo (CPU)				**71.630**			87.557			127.168					

Table 10: Comparative results of MS and MWU for the transaction cost function (b).

			MWU			MWU+LB			MS			MWU vs. MS		MWU+LB vs. MS	
Instance	σ	n	objective	CPU	$\|x^*\|_0$	objective	CPU	$\|x^*\|_0$	objective	CPU	$\|x^*\|_0$	Γ	Λ	Γ	Λ
port-orlib1_000	0.0047755010	31	**-0.0004356**	**1.024**	**1**	-0.0032985	3.179	10	-0.0039929	5.554	10	816.628%	5.424	21.053%	1.747
port-orlib1_050	0.0021522075	31	**-0.0004219**	**1.104**	**2**	-0.0030836	3.956	10	-0.0039929	5.697	10	846.405%	5.160	29.487%	1.440
port-orlib1_100	0.0010585969	31	**-0.0021266**	**1.455**	**8**	**-0.0021266**	2.363	**8**	-0.0041346	5.680	9	94.427%	3.904	94.427%	2.404
port-orlib1_150	0.0007158421	31	-0.0037476	**2.611**	7	**-0.0034443**	3.960	**5**	-0.0038742	3.893	7	3.377%	1.491	12.483%	0.983
port-orlib2_000	0.0028352430	85	-0.0001446	9.993	5	-0.0000687	17.275	2	-0.0002016	32.149	6	39.408%	3.217	193.362%	1.861
port-orlib2_050	0.0004953237	85	-0.0000917	**7.269**	**3**	**-0.0001101**	13.670	5	-0.0002016	32.370	6	119.742%	4.453	83.167%	2.368
port-orlib2_100	0.0002704062	85	**-0.0005385**	**33.373**	**9**	**-0.0005385**	42.610	**9**	**-0.0005385**	64.918	**9**	-0.002%	1.945	-0.002%	1.524
port-orlib2_150	0.0001663200	85	**-0.0024857**	1322.320	**10**	**-0.0024857**	262.913	**10**	**-0.0024857**	**1115.010**	**10**	-0.001%	0.843	-0.001%	4.241
port-orlib3_000	0.0015166351	89	-0.0001214	**7.340**	3	**-0.0001205**	21.984	**1**	-0.0001825	41.263	4	50.339%	5.622	51.466%	1.877
port-orlib3_050	0.0005849758	89	**-0.0001355**	**6.643**	**3**	**-0.0001355**	13.929	**3**	-0.0001825	37.436	4	34.663%	5.635	34.662%	2.688
port-orlib3_100	0.0003215941	89	**-0.0007110**	**17.459**	**6**	-0.0007768	50.735	7	-0.0007760	42.141	7	9.139%	2.414	-0.108%	0.831
port-orlib3_150	0.0002239117	89	-0.0032757	13349.900	**10**	-0.0032221	10905.202	**10**	**-0.0031912**	**2957.484**	**10**	-2.580%	0.222	-0.960%	0.271
port-orlib4_000	0.0029387241	98	-0.0001820	**11.705**	**3**	-0.0005766	16.740	7	**-0.0001414**	28.049	**3**	-22.293%	2.396	-75.477%	1.676
port-orlib4_050	0.0006828450	98	**-0.0001404**	**14.638**	**3**	-0.0003466	16.504	5	-0.0001414	26.859	**3**	0.719%	1.835	-59.208%	1.627
port-orlib4_100	0.0003059553	98	-0.0006288	**19.624**	**6**	-0.0006801	64.904	**6**	**-0.0006227**	33.050	**6**	-0.967%	1.684	-8.441%	0.509
port-orlib4_150	0.0001613979	98	**-0.0022239**	8371.655	**10**	**-0.0022239**	**6198.304**	**10**	**-0.0022239**	1570.873	**10**	0.002%	0.188	0.002%	0.253
port-orlib5_000	0.0016485224	225	**-0.0000647**	**72.465**	**7**	-0.0000712	149.351	8	-0.0000757	167.038	9	16.913%	2.305	6.265%	1.118
port-orlib5_050	0.0005150277	225	-0.0004978	**108.452**	8	-0.0004978	224.882	8	**-0.0001212**	291.879	**5**	-75.654%	2.691	-75.654%	1.298
port-orlib5_100	0.0003918260	225	-0.0013865	129.598	8	-0.0007129	**95.863**	8	**-0.0004068**	209.771	**6**	-70.660%	1.619	-42.937%	2.188
port-orlib5_150	0.0003272876	225	-0.0016889	180.181	**8**	-0.0016889	**169.206**	**8**	**-0.0010823**	546.634	**8**	-35.917%	3.034	-35.917%	3.231
$\sum$			**-0.0210489**	23668.809	**120**	-0.0262090	18277.530	140	-0.0285696	**7217.748**	142	**1823.689%**	**56.082**	**227.670%**	**34.135**
avg			**-0.0010524**	1183.440	**6.000**	-0.0013104	913.877	7.000	-0.0014285	**360.887**	7.100	**91.184%**	**2.804**	**11.383%**	**1.707**
std			(0.0011464)	(3419.072)	(2.920)	(0.0012441)	(2722.510)	(2.753)	(0.0015780)	(737.085)	(2.469)	(257.358%)	(1.725)	(62.458%)	(0.985)
geo (CPU)				**31.288**			47.339			65.661					

Table 11: Comparative results of MS and MWU for the transaction cost function (c).

			MWU			MWU+LB			MS			MWU vs. MS		MWU+LB vs. MS	
Instance	σ	n	objective	CPU	$\|x^*\|_0$	objective	CPU	$\|x^*\|_0$	objective	CPU	$\|x^*\|_0$	Γ	Λ	Γ	Λ
port-orlib1_000	0.0047755010	31	**-0.0003929**	**0.825**	**2**	-0.0005442	3.319	9	-0.0008732	5.750	9	122.260%	6.970	60.446%	1.732
port-orlib1_050	0.0021522075	31	**-0.0003264**	**1.040**	**3**	-0.0006592	3.127	6	-0.0008732	6.008	9	167.496%	5.777	32.464%	1.921
port-orlib1_100	0.0010585969	31	-0.0007308	**1.780**	**9**	**-0.0006770**	4.030	**9**	-0.0010003	5.716	**9**	36.887%	3.211	47.760%	1.418
port-orlib1_150	0.0007158421	31	-0.0027051	**2.165**	**5**	-0.0016986	4.537	6	**-0.0010005**	3.970	8	-63.015%	1.834	-41.098%	0.875
port-orlib2_000	0.0028352430	85	-0.0000428	**4.970**	6	**-0.0000004**	13.170	**3**	-0.0000428	28.166	6	0.000%	5.667	11034.235%	2.139
port-orlib2_050	0.0004953237	85	-0.0000784	**4.272**	9	-0.0000544	16.765	7	**-0.0000464**	31.663	**6**	-40.774%	7.412	-14.598%	1.889
port-orlib2_100	0.0002704062	85	-0.0001552	**16.052**	**9**	-0.0001552	33.207	**9**	**-0.0001273**	96.677	**9**	-17.953%	6.023	-17.956%	2.911
port-orlib2_150	0.0001663200	85	-0.0007034	**165.865**	**10**	-0.0008274	3627.871	**10**	**-0.0007033**	327.515	**10**	-0.001%	1.975	-14.993%	0.090
port-orlib3_000	0.0015166351	89	-0.0001257	**6.590**	**1**	-0.0001605	13.743	9	**-0.0001232**	29.997	6	-1.977%	4.552	-23.212%	2.183
port-orlib3_050	0.0005849758	89	-0.0001166	**10.815**	**6**	**-0.0001025**	20.975	**6**	-0.0001232	33.973	**6**	5.667%	3.141	20.243%	1.620
port-orlib3_100	0.0003215941	89	**-0.0002009**	**22.068**	**7**	-0.0004476	29.281	**7**	**-0.0002009**	180.959	7	0.000%	8.200	-55.126%	6.180
port-orlib3_150	0.0002239117	89	**-0.0007063**	9872.064	**10**	-0.0009387	9067.600	**10**	**-0.0007063**	**775.900**	**10**	0.000%	0.079	-24.759%	0.086
port-orlib4_000	0.0029387241	98	**-0.0000210**	**8.926**	**2**	-0.0001277	23.203	7	-0.0001137	34.462	5	441.257%	3.861	-10.950%	1.485
port-orlib4_050	0.0006828450	98	**-0.0000737**	**9.197**	**3**	-0.0001134	24.853	6	-0.0001578	35.712	8	114.188%	3.883	39.146%	1.437
port-orlib4_100	0.0003059553	98	**-0.0002065**	**10.295**	9	-0.0002681	23.826	**7**	**-0.0002065**	36.192	9	0.000%	3.515	-22.957%	1.519
port-orlib4_150	0.0001613979	98	-0.0006690	**6321.129**	**9**	**-0.0005819**	3312.971	10	-0.0006155	4837.097	10	-8.008%	0.765	5.764%	1.460
port-orlib5_000	0.0016485224	225	-0.0000198	**93.510**	**9**	-0.0000172	124.442	**9**	**-0.0000155**	151.180	**9**	-21.759%	1.617	-9.970%	1.215
port-orlib5_050	0.0005150277	225	-0.0001077	823.956	8	-0.0000869	**257.904**	**7**	**-0.0000762**	1347.551	**7**	-29.266%	1.635	-12.290%	5.225
port-orlib5_100	0.0003918260	225	-0.0002729	**130.751**	7	-0.0006698	148.388	**6**	**-0.0001507**	638.984	**6**	-44.768%	4.887	-77.499%	4.306
port-orlib5_150	0.0003272876	225	**-0.0003806**	**104.124**	**8**	**-0.0003806**	142.845	**8**	-0.0002542	269.757	**8**	-33.199%	2.591	-33.199%	1.888
$\sum$			-0.0080357	17610.394	**132**	-0.0085112	16896.057	151	**-0.0074109**	**8877.229**	157	**627.036%**	**77.594**	**10881.450%**	**41.580**
avg			-0.0004018	880.520	**6.600**	-0.0004256	844.803	7.550	**-0.0003705**	**443.861**	7.850	**31.352%**	**3.880**	**544.073%**	**2.079**
std			(0.0005941)	(2540.745)	(2.945)	(0.0004182)	(2203.036)	(1.820)	(0.0003567)	(1089.089)	(1.599)	(113.077%)	(2.276)	(2469.367%)	1.536
geo (CPU)				**25.737**			50.570			73.957					

Table 12: Comparative results of MS and MWU for the transaction cost function (d).

			MWU			MWU+LB			MS			MWU vs. MS		MWU+LB vs. MS	
Instance	σ	n	objective	CPU	$\|x^*\|_0$	objective	CPU	$\|x^*\|_0$	objective	CPU	$\|x^*\|_0$	Γ	Λ	Γ	Λ
port-orlib1_000	0.0047755010	31	0.0217170	**1.877**	**6**	0.0395563	14.952	10	**0.0487432**	43.180	10	-124.447%	23.005	-23.225%	2.888
port-orlib1_050	0.0021522075	31	0.0469337	**3.138**	**10**	0.0476659	56.161	**10**	**0.0488171**	37.351	**10**	-4.013%	11.903	-2.415%	0.665
port-orlib1_100	0.0010585969	31	0.0432059	**2.984**	**10**	0.0413710	4.962	**10**	**0.0498939**	58.515	**10**	-15.479%	19.610	-20.601%	11.793
port-orlib1_150	0.0007158421	31	**0.0399172**	**2.588**	**9**	0.0376060	6.172	**9**	0.0388779	6.757	**9**	2.604%	2.611	-3.382%	1.095
port-orlib2_000	0.0028352430	85	0.0136601	10.510	4	0.0133721	**8.726**	**3**	**0.0184047**	375.738	10	-34.734%	35.751	-37.635%	43.060
port-orlib2_050	0.0004953237	85	0.0383876	39.267	7	**0.0387789**	**33.822**	**6**	0.0186861	522.376	10	51.323%	13.303	51.814%	15.445
port-orlib2_100	0.0002704062	85	**0.0283756**	4226.211	**10**	0.0166890	4382.785	**10**	0.0207083	**524.517**	**10**	27.021%	0.124	-24.083%	0.120
port-orlib2_150	0.0001663200	85	**0.0315010**	3642.877	**10**	0.0307374	3989.962	**10**	0.0268662	**1599.415**	**10**	14.713%	0.439	12.594%	0.401
port-orlib3_000	0.0015166351	89	0.0116564	**9.548**	**2**	0.0029338	22.035	8	**0.0234077**	240.592	10	-100.814%	25.198	-697.857%	10.919
port-orlib3_050	0.0005849758	89	0.0347923	**19.026**	**8**	**0.0348732**	102.802	**8**	0.0239912	441.404	10	31.044%	23.200	31.204%	4.294
port-orlib3_100	0.0003215941	89	0.0342867	903.586	**9**	**0.0392775**	2142.790	10	0.0188460	**719.657**	10	45.034%	0.796	52.018%	0.336
port-orlib3_150	0.0002239117	89	**0.0284570**	15650.882	**10**	0.0269692	11231.379	**10**	0.0255798	**5050.864**	**10**	10.111%	0.323	5.152%	0.450
port-orlib4_000	0.0029387241	98	0.0130890	**4.641**	**2**	0.0123611	66.034	10	**0.0233697**	432.908	10	-78.545%	93.279	-89.058%	6.556
port-orlib4_050	0.0006828450	98	**0.0583659**	**32.832**	**10**	0.0541541	91.192	**10**	0.0239543	272.405	**10**	58.958%	8.297	55.766%	2.987
port-orlib4_100	0.0003059553	98	0.0135301	1082.307	**10**	0.0078843	696.938	**10**	**0.0248876**	**513.718**	**10**	-83.943%	0.475	-215.660%	0.737
port-orlib4_150	0.0001613979	98	0.0236738	15929.002	**10**	**0.0260514**	24604.167	**10**	0.0252765	**5982.823**	**10**	-6.770%	0.376	2.974%	0.243
port-orlib5_000	0.0016485224	225	0.0326302	1448.934	**6**	**0.032836**	**788.687**	**6**	0.0017349	5513.344	10	94.683%	3.805	94.717%	6.991
port-orlib5_050	0.0005150277	225	0.0031309	**302.041**	**9**	**0.0073133**	370.425	10	0.0018351	350.660	**9**	41.386%	1.161	74.907%	0.947
port-orlib5_100	0.0003918260	225	0.0078564	5516.043	**7**	0.0035668	4774.516	8	**0.0103946**	**1525.448**	10	-32.308%	0.277	-191.430%	0.319
port-orlib5_150	0.0003272876	225	**0.0143170**	968.684	**10**	0.0143169	**730.436**	**10**	**0.0143170**	913.920	**10**	0.000%	0.943	0.000%	1.251
$\sum$			**0.5394835**	49796.978	**159**	0.5283151	54118.943	178	0.4885919	**25125.592**	198	-104.177%	**264.875**	-924.200%	**111.494**
avg			**0.0269742**	2489.849	**7.950**	0.0264158	2705.947	8.900	0.0244296	**1256.280**	9.900	-5.209%	**13.244**	-46.210%	**5.575**
std			(0.0145719)	(4819.775)	(2.685)	(0.0155312)	(5846.227)	(1.917)	(0.0135588)	(1891.435)	(0.308)	(56.991%)	(21.729)	(172.079%)	9.900
geo (CPU)				**135.816**			247.786			426.464					

Table 13: Comparative results of MS and MWU for the transaction cost function (e).