BoxPacker Documentation

Release version 4

Doug Wright

Apr 01, 2024

CONTENTS

BoxPacker is an implementation of the "4D" bin packing/knapsack problem i.e. given a list of items, how many boxes do you need to fit them all in.

Especially useful for e.g. e-commerce contexts when you need to know box size/weight to calculate shipping costs, or even just want to know the right number of labels to print.

CHAPTER

ONE

LICENSE

BoxPacker is licensed under the MIT license.

1.1 Installation

The recommended way to install BoxPacker is to use Composer. From the command line simply execute the following to add `dvdoug/boxpacker` to your project's `composer.json` file. Composer will automatically take care of downloading the source and configuring an autoloader:

```
composer require dvdoug/boxpacker
```

If you don't want to use Composer, the code is available to download from GitHub

1.1.1 Requirements

BoxPacker v4 is compatible with PHP 8.2+

1.1.2 Versioning

BoxPacker follows Semantic Versioning. For details about differences between releases please see What's new

1.2 Principles of operation

Bin packing is an NP-hard problem and there is no way to always achieve an optimum solution without running through every single permutation. But that's OK because this implementation is designed to simulate a naive human approach to the problem rather than search for the "perfect" solution.

This is for 2 reasons:

1. It's quicker
2. It doesn't require the person actually packing the box to be given a 3D diagram explaining just how the items are supposed to fit.

At a high level, the algorithm works like this:

- Pack largest (by volume) items first
- Pack vertically up the side of the box
- Pack side-by-side where item under consideration fits alongside the previous item

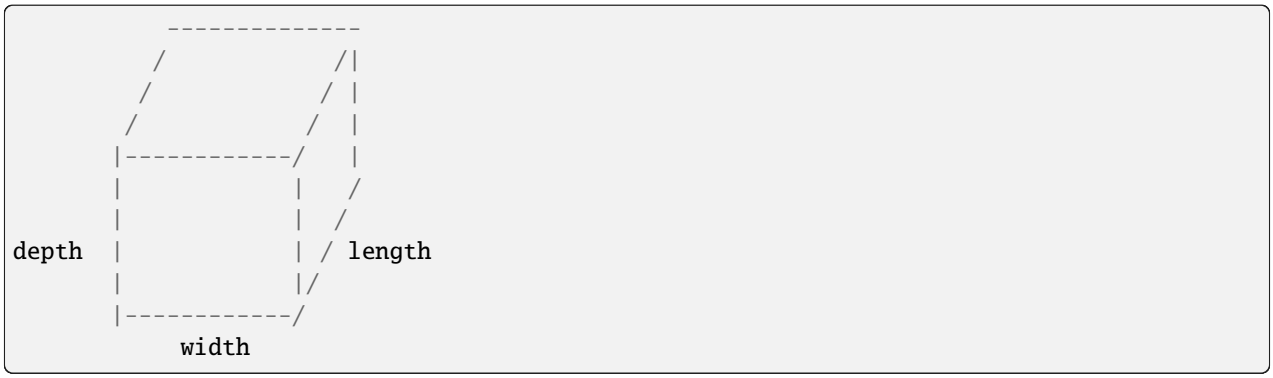
- If more than 1 box is needed to accommodate all of the items, then aim for boxes of roughly equal weight (e.g. 3 medium size/weight boxes are better than 1 small light box and 2 that are large and heavy)

1.3 Getting started

BoxPacker is designed to integrate as seamlessly as possible into your existing systems, and therefore makes strong use of PHP interfaces. Applications wanting to use this library will typically already have PHP domain objects/entities representing the items needing packing, so BoxPacker attempts to take advantage of these as much as possible by allowing you to pass them directly into the Packer rather than needing you to construct library-specific datastructures first. This also makes it much easier to work with the output of the Packer - the returned list of packed items in each box will contain your own objects, not simply references to them so if you want to calculate value for insurance purposes or anything else this is easy to do.

Similarly, although it's much more uncommon to already have 'Box' objects before implementing this library, you'll typically want to implement them in an application-specific way to allow for storage/retrieval from a database. The Packer also allows you to pass in these objects directly too.

To accommodate the wide variety of possible object types, the library defines two interfaces `BoxPacker\Item` and `BoxPacker\Box` which define methods for retrieving the required dimensional data - e.g. `getWidth()`. There's a good chance you may already have at least some of these defined.



If you do happen to have methods defined with those names already, **and they are incompatible with the interface expectations**, then this will be only case where some kind of wrapper object would be needed.

1.3.1 Examples

Packing a set of items into a given set of box types

```
<?php
    use DVDoug\BoxPacker\Rotation;
    use DVDoug\BoxPacker\Packer;
    use DVDoug\BoxPacker\Test\TestBox;  // use your own `Box` implementation
    use DVDoug\BoxPacker\Test\TestItem; // use your own `Item` implementation

    $packer = new Packer();

    /*
     * Add choices of box type - in this example the dimensions are passed in directly␣
↪via constructor,
```

(continues on next page)

(continued from previous page)

```
     * but for real code you would probably pass in objects retrieved from a database␣
↪instead
     */
    $packer->addBox(
        new TestBox(
            reference: 'Le petite box',
            outerWidth: 300,
            outerLength: 300,
            outerDepth: 10,
            emptyWeight: 10,
            innerWidth: 296,
            innerLength: 296,
            innerDepth: 8,
            maxWeight: 1000
        )
    );
    $packer->addBox(
        new TestBox(
            reference: 'Le grande box',
            outerWidth: 3000,
            outerLength: 3000,
            outerDepth: 100,
            emptyWeight: 100,
            innerWidth: 2960,
            innerLength: 2960,
            innerDepth: 80,
            maxWeight: 10000
        )
    );

    /*
     * Add items to be packed - e.g. from shopping cart stored in user session. Again,␣
↪the dimensional information
     * (and keep-flat requirement) would normally come from a DB
     */
    $packer->addItem(
        item: new TestItem(
            description: 'Item 1',
            width: 250,
            length: 250,
            depth: 12,
            weight: 200,
            allowedRotation: Rotation::KeepFlat
        ),
        qty: 1
    );
    $packer->addItem(
        item: new TestItem(
            description: 'Item 2',
            width: 250,
            length: 250,
            depth: 12,
```

(continues on next page)

(continued from previous page)

```
            weight: 200,
            allowedRotation: Rotation::KeepFlat
        ),
        qty: 2
    );
    $packer->addItem(
        item: new TestItem(
            description: 'Item 3',
            width: 250,
            length: 250,
            depth: 24,
            weight: 200,
            allowedRotation: Rotation::BestFit
        ),
        qty: 1
    );

    $packedBoxes = $packer->pack();

    echo "These items fitted into " . count($packedBoxes) . " box(es)" . PHP_EOL;
    foreach ($packedBoxes as $packedBox) {
        $boxType = $packedBox->getBox(); // your own box object, in this case TestBox
        echo "This box is a {$boxType->getReference()}, it is {$boxType->getOuterWidth()}
↪mm wide, {$boxType->getOuterLength()}mm long and {$boxType->getOuterDepth()}mm high" .␣
↪PHP_EOL;
        echo "The combined weight of this box and the items inside it is {$packedBox->
↪getWeight()}g" . PHP_EOL;

        echo "The items in this box are:" . PHP_EOL;
        $packedItems = $packedBox->items;
        foreach ($packedItems as $packedItem) { // $packedItem->item is your own item␣
↪object, in this case TestItem
            echo $packedItem->item->getDescription() . PHP_EOL;
        }
    }
```

Does a set of items fit into a particular box

```
<?php
    use DVDoug\BoxPacker\Rotation;
    use DVDoug\BoxPacker\Packer;
    use DVDoug\BoxPacker\ItemList;
    use DVDoug\BoxPacker\VolumePacker;
    use DVDoug\BoxPacker\Test\TestBox;  // use your own `Box` implementation
    use DVDoug\BoxPacker\Test\TestItem; // use your own `Item` implementation

    /*
     * To just see if a selection of items will fit into one specific box
     */
    $box = new TestBox(
```

(continues on next page)

(continued from previous page)

```
        reference: 'Le box',
        outerWidth: 300,
        outerLength: 300,
        outerDepth: 10,
        emptyWeight: 10,
        innerWidth: 296,
        innerLength: 296,
        innerDepth: 8,
        maxWeight: 1000
    );

    $items = new ItemList();
    $items->insert(
        new TestItem(
            description: 'Item 1',
            width: 297,
            length: 296,
            depth: 2,
            weight: 200,
            allowedRotation: Rotation::BestFit
        )
    );
    $items->insert(
        new TestItem(
            description: 'Item 2',
            width: 297,
            length: 296,
            depth: 2,
            weight: 500,
            allowedRotation: Rotation::BestFit
        )
    );
    $items->insert(
        new TestItem(
            description: 'Item 3',
            width: 296,
            length: 296,
            depth: 4,
            weight: 290,
            allowedRotation: Rotation::BestFit
        )
    );

    $volumePacker = new VolumePacker($box, $items);
    $packedBox = $volumePacker->pack(); //$packedBox->items contains the items that fit

    echo "The items in this box are:" . PHP_EOL;
    $packedItems = $packedBox->items;
    foreach ($packedItems as $packedItem) { // $packedItem->getItem() is your own item␣
↪object, in this case TestItem
        echo $packedItem->item->getDescription() . PHP_EOL;
    }
```

1.4 Rotation

1.4.1 Items

BoxPacker gives you full control of how (or if) an individual item may be rotated to fit into a box, controlled via the `getAllowedRotation()` method on the `BoxPacker\Item` interface.

Best fit

To allow an item to be placed in any orientation.

```
<?php
    use DVDoug\BoxPacker\Item;
    use DVDoug\BoxPacker\Rotation;

    class YourItem implements Item
    {
        public function getAllowedRotation(): Rotation
        {
            return Rotation::BestFit;
        }
    }
```

Keep flat

For items that must be shipped "flat" or "this way up".

```
<?php
    use DVDoug\BoxPacker\Item;
    use DVDoug\BoxPacker\Rotation;

    class YourItem implements Item
    {
        public function getAllowedRotation(): Rotation
        {
            return Rotation::KeepFlat;
        }
    }
```

No rotation

It is also possible to stop an item from being rotated at all. This is not normally useful for ecommerce, but can be useful when trying to use the library in other contexts e.g. packing sprites.

```
<?php
    use DVDoug\BoxPacker\Item;
    use DVDoug\BoxPacker\Rotation;

    class YourItem implements Item
    {
```

(continues on next page)

(continued from previous page)

```
        public function getAllowedRotation(): Rotation
        {
            return Rotation::Never;
        }
    }
```

1.4.2 Boxes

BoxPacker operates internally by positioning items in "rows", firstly by placing items across the width of the box, then when there is no more space starting a new row further along the length.

However, due to the nature of the placement heuristics, better packing is sometimes achieved by going the other way i.e. placing items along the length first. By default BoxPacker handles this by trying packing both ways around, transposing widths and lengths as appropriate.

For most purposes this is fine, when the boxes come to be packed in real life it is done via the top and the direction used for placement doesn't matter. However, sometimes the "box" being given to BoxPacker is actually a truck or other side-loaded container and in these cases it is sometimes desirable to enforce the packing direction.

This can be done when using the `VolumePacker` by calling the `packAcrossWidthOnly` method.

```
<?php
use DVDoug\BoxPacker\VolumePacker;

$volumePacker = new VolumePacker($box, $items);
$volumePacker->packAcrossWidthOnly();
$packedBox = $volumePacker->pack();
```

1.5 Sortation

BoxPacker (mostly) uses "online" algorithms, that is it packs sequentially, with no regard for what comes next. Therefore the order of items, or the order of boxes are of crucial importance in obtaining good results.

By default, BoxPacker will try to be as smart as possible about this, packing larger/heavier items into the bottom of a box, with smaller/lighter items that might get crushed placed above them. It will also prefer to use smaller boxes where possible, rather than larger ones.

However, BoxPacker also allows you to influence many of these decisions if you prefer.

1.5.1 Items

You may wish to explicitly pack heavier items before larger ones. Or larger ones before heavier ones. Or prefer to keep items of a similar "group" together (whatever that might mean for your application). The `ItemList` class supports this via two methods.

Supplying a pre-sorted list

If you already have your items in a pre-sorted array (e.g. when using a database `ORDER BY`, you can construct an `ItemList` directly from it. You can also use this mechanism if you know that all of your items have identical dimensions and therefore having BoxPacker sort them before commencing packing would just be a waste of CPU time.

```
$itemList = ItemList::fromArray($anArrayOfItems, true); // set the 2nd param to true if␣
↪presorted
```

Overriding the default algorithm

First, create your own implementation of the `ItemSorter` interface implementing your particular requirements:

```
/**
 * A callback to be used with usort(), implementing logic to determine which Item is a␣
↪higher priority for packing.
 */
YourApplicationItemSorter implements DVDoug\BoxPacker\ItemSorter
{
    /**
     * Return -1 if $itemA is preferred, 1 if $itemB is preferred or 0 if neither is␣
↪preferred.
     */
    public function compare(Item $itemA, Item $itemB): int
    {
        // your logic to determine ordering goes here. Remember, that Item is your own␣
↪object,
        // and you have full access to all methods on it, not just the ones from the␣
↪Item interface
    }
}
```

Then, pass this to the `ItemList` constructor

```
$sorter = new YourApplicationItemSorter();
$itemList = new ItemList($sorter);
```

Enforcing strict ordering

Regardless of which of the above methods you use, BoxPacker's normal mode of operation is to respect the sort ordering *but not at the expense of packing density*. If an item in the list is too large to fit into a particular space, BoxPacker will temporarily skip over it and will try the next item in the list instead.

This typically works well for ecommerce, but in some applications you may want your custom sort to be absolutely determinative. You can do this by calling `beStrictAboutItemOrdering()`.

```
$packer = new Packer();
$packer->beStrictAboutItemOrdering(true); // or false to turn strict ordering off again

$volumePacker = new VolumePacker(...);
$volumePacker->beStrictAboutItemOrdering(true); // or false to turn strict ordering off␣
↪again
```

1.5.2 Box types

BoxPacker's default algorithm assumes that box size/weight is a proxy for cost and therefore seeks to use the smallest/lightest type of box possible for a set of items. However in some cases this assumption might not be true, or you may have alternate reasons for preferring to use one type of box over another. The `BoxList` class supports this kind of application-controlled sorting via two methods.

Supplying a pre-sorted list

If you already have your items in a pre-sorted array (e.g. when using a database `ORDER BY`, you can construct an `BoxList` directly from it.

```
$boxList = BoxList::fromArray($anArrayOfBoxes, true); // set the 2nd param to true if␣
↪presorted
```

Overriding the default algorithm

First, create your own implementation of the `BoxSorter` interface implementing your particular requirements:

```
/**
 * A callback to be used with usort(), implementing logic to determine which Box is
↪"better".
 */
YourApplicationBoxSorter implements DVDoug\BoxPacker\BoxSorter
{
    /**
     * Return -1 if $boxA is "best", 1 if $boxB is "best" or 0 if neither is "best".
     */
    public function compare(Box $boxA, Box $boxB): int
    {
        // your logic to determine ordering goes here. Remember, that Box is your own␣
↪object,
        // and you have full access to all methods on it, not just the ones from the Box␣
↪interface
    }
}
```

Then, pass this to the `BoxList` constructor

```
$sorter = new YourApplicationBoxSorter();
$boxList = new BoxList($sorter);
```

1.5.3 Choosing between permutations

In a scenario where even the largest box type is not large enough to contain all of the items, BoxPacker needs to decide which is the "best" possible first box, so it can then pack the remaining items into a second box (and so on). If there are two different box types that each hold the same number of items (but different items), which one should be picked? What if one of the boxes can hold an additional item, but is twice as large? Is it better to minimise the number of boxes, or their volume?

By default, BoxPacker will optimise for the largest number of items in a box, with volume acting as a tie-breaker. This can also be changed:

Overriding the default algorithm

First, create your own implementation of the `PackedBoxSorter` interface implementing your particular requirements:

```
/**
 * A callback to be used with usort(), implementing logic to determine which PackedBox␣
↪is "better".
 */
YourApplicationPackedBoxSorter implements DVDoug\BoxPacker\PackedBoxSorter
{
    /**
     * Return -1 if $boxA is "best", 1 if $boxB is "best" or 0 if neither is "best".
     */
    public function compare(PackedBox $boxA, PackedBox $boxB): int
    {
        // your logic to determine "best" goes here
    }
}
```

Then, pass this to the `Packer`

```
$sorter = new YourApplicationPackedBoxSorter();

$packer = new Packer();
$packer->setPackedBoxSorter($sorter);
```

1.6 Weight distribution

If you are shipping a large number of items to a single customer as many businesses do, it might be that more than one box is required to accommodate all of the items. A common scenario which you'll have probably encountered when receiving your own deliveries is that the first box(es) will be absolutely full as the warehouse operative will have tried to fit in as much as possible. The last box by comparison will be virtually empty and mostly filled with protective inner packing.

There's nothing intrinsically wrong with this, but it can be a bit annoying for e.g. couriers and customers to receive e.g. a 20kg box which requires heavy lifting alongside a similarly sized box that weighs hardly anything at all. If you have to send two boxes anyway, it would be much better in such a situation to have e.g. an 11kg box and a 10kg box instead.

Happily, this smoothing out of weight is handled automatically for you by BoxPacker - once the initial dimension-only packing is completed, a second pass is made that reallocates items from heavier boxes into any lighter ones that have space.

For most use-cases the benefits are worth the extra computation time - however if a single "packing" for your scenarios involves a very large number of permutations e.g. thousands of items, you may wish to tune this behaviour.

By default, the weight distribution pass is made whenever the items fit into 12 boxes or less. To reduce (or increase) the threshold, call `setMaxBoxesToBalanceWeight()`

```
<?php
    use DVDoug\BoxPacker\Packer;

    $packer = new Packer();
    $packer->setMaxBoxesToBalanceWeight(3);
```

Note: A threshold value of either 0 or 1 will disable the weight distribution pass completely

1.7 Too-large items

As a library, by default BoxPacker makes the design choice that any errors or exceptions thrown during operation are best handled by you and your own code as the appropriate way to to handle a failure will vary from application to application. There is no attempt made to handle/recover from them internally.

This includes the case where there are no boxes large enough to pack a particular item. The normal operation of the Packer class is to throw an `NoBoxesAvailableException`. If your application has well-defined logging and monitoring it may be sufficient to simply allow the exception to bubble up to your generic handling layer and handle like any other runtime failure. Applications that do that can make an assumption that if no exceptions were thrown, then all items were successfully placed into a box.

Alternatively, you might wish to catch the exception explicitly and have domain-specific handling logic e.g.

```
<?php
    use DVDoug\BoxPacker\NoBoxesAvailableException;
    use DVDoug\BoxPacker\Packer;
    use DVDoug\BoxPacker\Test\TestBox;  // use your own `Box` implementation
    use DVDoug\BoxPacker\Test\TestItem; // use your own `Item` implementation

    try {
        $packer = new Packer();

        $packer->addBox(new TestBox('Le petite box', 300, 300, 10, 10, 296, 296, 8,␣
↪1000));
        $packer->addBox(new TestBox('Le grande box', 3000, 3000, 100, 100, 2960, 2960,␣
↪80, 10000));

        $packer->addItem(new TestItem('Item 1', 2500, 2500, 20, 2000,␣
↪Rotation::BestFit));
        $packer->addItem(new TestItem('Item 2', 25000, 2500, 20, 2000,␣
↪Rotation::BestFit));
        $packer->addItem(new TestItem('Item 3', 2500, 2500, 20, 2000,␣
↪Rotation::BestFit));

        $packedBoxes = $packer->pack();
    } catch (NoBoxesAvailableException $e) {
```

(continues on next page)

(continued from previous page)

```
        $affectedItems = $e->getAffectedItems(); // you can retrieve the list of␣
↪affected items

        //add code to pause dispatch, page someone and/or perform any other handling of␣
↪your choice
    }
```

However, an `Exception` is for exceptional situations and for some businesses, some items being too large and thus requiring special handling might be considered a normal everyday situation. For these applications, having an `Exception` thrown which interrupts execution might be not be wanted or be considered problematic.

BoxPacker also supports this type of workflow by setting `throwOnUnpackableItem` to `false`. When doing this, the return value of `->pack()` is a `PackedBoxList` as in regular operation. You can then call `->getUnpackedItems()` to retrieve the list of those that were not able to be packed. An example:

```
<?php
    use DVDoug\BoxPacker\Packer;
    use DVDoug\BoxPacker\Test\TestBox;  // use your own `Box` implementation
    use DVDoug\BoxPacker\Test\TestItem; // use your own `Item` implementation

    $packer = new Packer();
    $packer->throwOnUnpackableItem(false); // defaults to true

    $packer->addBox(new TestBox('Le petite box', 300, 300, 10, 10, 296, 296, 8, 1000));
    $packer->addBox(new TestBox('Le grande box', 3000, 3000, 100, 100, 2960, 2960, 80,␣
↪10000));

    $packer->addItem(new TestItem('Item 1', 2500, 2500, 20, 2000, Rotation::BestFit));
    $packer->addItem(new TestItem('Item 2', 25000, 2500, 20, 2000, Rotation::BestFit));
    $packer->addItem(new TestItem('Item 3', 2500, 2500, 20, 2000, Rotation::BestFit));

    $packedBoxes = $packer->pack();

    // It is *very* important to check this is an empty list (or not) when exceptions␣
↪are disabled!
    $unpackedItems = $packer->getUnpackedItems();
```

1.8 Positional information

It is possible to see the precise positional and dimensional information of each item as packed. This is exposed as x,y,z co-ordinates from origin, alongside width/length/depth in the packed orientation.

```
z    y
|   /
|  /
| /
|/
0------x
```

Example:

```php
<?php

    // assuming packing already took place
    foreach ($packedBoxes as $packedBox) {
        $packedItems = $packedBox->items;
        foreach ($packedItems as $packedItem) { // $packedItem->item is your own item␣
↪object
            echo $packedItem->item->getDescription() .  ' was packed at co-ordinate ' ;
            echo '(' . $packedItem->x . ', ' . $packedItem->y . ', ' . $packedItem->z .
↪') with ';
            echo 'w' . $packedItem->width . ', l' . $packedItem->length . ', d' .
↪$packedItem->depth;
            echo PHP_EOL;
        }
    }
```

A visualiser is also available.

1.9 Limited supply boxes

In standard/basic use, BoxPacker will assume you have an adequate enough supply of each box type on hand to cover all eventualities i.e. your warehouse will be very well stocked and the concept of "running low" is not applicable.

However, if you only have limited quantities of boxes available and you have accurate stock control information, you can feed this information into BoxPacker which will then take it into account so that it won't suggest a packing which would take you into negative stock.

To do this, have your box objects implement the `BoxPacker\LimitedSupplyBox` interface which has a single additional method over the standard `BoxPacker\Box` namely `getQuantityAvailable()`. The library will automatically detect this and use the information accordingly.

1.10 Custom constraints

For more advanced use cases where greater control over the contents of each box is required (e.g. legal limits on the number of hazardous items per box, or perhaps fragile items requiring an extra-strong outer box) you may implement the `BoxPacker\ConstrainedPlacementItem` interface which contains an additional callback method allowing you to decide whether to allow an item may be packed into a box or not.

As with all other library methods, the objects passed into this callback are your own - you have access to their full range of properties and methods to use when evaluating a constraint, not only those defined by the standard `BoxPacker\Item` interface.

1.10.1 Example - only allow 2 batteries per box

```php
<?php
    use DVDoug\BoxPacker\PackedBox;

    class LithiumBattery implements ConstrainedPlacementItem
    {

        /**
         * Max 2 batteries per box.
         *
         * @param  PackedBox      $packedBox
         * @param  int            $proposedX
         * @param  int            $proposedY
         * @param  int            $proposedZ
         * @param  int            $width
         * @param  int            $length
         * @param  int            $depth
         * @return bool
         */
        public function canBePacked(
            PackedBox $packedBox,
            int $proposedX,
            int $proposedY,
            int $proposedZ,
            int $width,
            int $length,
            int $depth
        ): bool {
            $batteriesPacked = 0;
            foreach ($packedBox->items as $packedItem) {
              if ($packedItem->item instanceof LithiumBattery) {
                  $batteriesPacked++;
              }
            }

            if ($batteriesPacked < 2) {
                return true;  // allowed to pack
            } else {
                return false; // 2 batteries already packed, no more allowed in this box
            }
        }
    }
```

1.10.2 Example - don't allow batteries to be stacked

```php
<?php
    use DVDoug\BoxPacker\PackedBox;
    use DVDoug\BoxPacker\PackedItem;

    class LithiumBattery implements ConstrainedPlacementItem
    {

        /**
         * Batteries cannot be stacked on top of each other.
         *
         * @param  PackedBox      $packedBox
         * @param  int            $proposedX
         * @param  int            $proposedY
         * @param  int            $proposedZ
         * @param  int            $width
         * @param  int            $length
         * @param  int            $depth
         * @return bool
         */
        public function canBePacked(
            PackedBox $packedBox,
            int $proposedX,
            int $proposedY,
            int $proposedZ,
            int $width,
            int $length,
            int $depth
        ): bool {
            $alreadyPackedType = array_filter(
                iterator_to_array($packedBox->items, false),
                function (PackedItem $item) {
                    return $item->item->getDescription() === 'Battery';
                }
            );

            /** @var PackedItem $alreadyPacked */
            foreach ($alreadyPackedType as $alreadyPacked) {
                if (
                    $alreadyPacked->z + $alreadyPacked->depth === $proposedZ &&
                    $proposedX >= $alreadyPacked->x && $proposedX <= ($alreadyPacked->x␣
↪+ $alreadyPacked->width) &&
                    $proposedY >= $alreadyPacked->y && $proposedY <= ($alreadyPacked->y␣
↪+ $alreadyPacked->length)) {
                    return false;
                }
            }

            return true;
        }
    }
```

1.11 Used / remaining space

After packing it is possible to see how much physical space in each `PackedBox` is taken up with items, and how much space was unused (air). This information might be useful to determine whether it would be useful to source alternative/additional sizes of box.

At a high level, the `getVolumeUtilisation()` method exists which calculates how full the box is as a percentage of volume.

Lower-level methods are also available for examining this data in detail either using `getUsed[Width|Length|Depth()]` (a hypothetical box placed around the items) or `getRemaining[Width|Length|Depth()]` (the difference between the dimensions of the actual box and the hypothetical box).

Note: BoxPacker will try to pack items into the smallest box available

1.11.1 Example - warning on a massively oversized box

```php
<?php

    // assuming packing already took place
    foreach ($packedBoxes as $packedBox) {
        if ($packedBox->getVolumeUtilisation() < 20) {
            // box is 80% air, log a warning
        }
    }
```

1.12 Permutations

Normally BoxPacker will try and make smart choice(s) of box when building a packing solution that minimise the both the number used and also their size (see *Sortation*). This is usually the most optimal arrangement from a logistical point of view, both for efficiency and for shipping cost. Supplying custom sorters as outlined on that page allow you to influence those decisions if needed.

You may however wish for BoxPacker to not make any of these decisions, but simply have it calculate all[1] of the possible combinations of box for you to then filter/select inside your own application. You can do this by calling `packAllPermutations()` on `Packer` instead of `pack()`:

```php
<?php
    use DVDoug\BoxPacker\Packer;
    use DVDoug\BoxPacker\Test\TestBox;  // use your own `Box` implementation
    use DVDoug\BoxPacker\Test\TestItem; // use your own `Item` implementation

    $packer = new Packer();

    $packer->addBox(new TestBox('Light box', 100, 100, 100, 1, 100, 100, 100, 100));
    $packer->addBox(new TestBox('Heavy box', 100, 100, 100, 100, 100, 100, 100, 10000));
```

(continues on next page)

[1] "all" refers to the permutations of boxes e.g. 1×large OR 2×medium OR 1×medium + 2×small etc. It does not refer to any permutations of items within, since that level of combinatorial explosion would be completely unmanageable

(continued from previous page)

```
$packer->addItem(new TestItem('Item 1', 100, 100, 100, 75, Rotation::BestFit));
$packer->addItem(new TestItem('Item 2', 100, 100, 100, 75, Rotation::BestFit));
$packer->addItem(new TestItem('Item 3', 100, 100, 100, 75, Rotation::BestFit));

$permutations = $packer->packAllPermutations(); // an array of PackedBoxList objects
```

Warning: Although the regular `pack()` does evaluate multiple permutations when calculating its result, it is also able to use various optimisation techniques to reduce this to a minimum. By definition, `packAllPermutations()` cannot take advantage of these, and the number of permutations can easily become **very large** with corresponding impacts on runtime. Be absolutely sure you want to use this method, rather than use a custom sorter.

1.13 Changelog

1.13.1 4.x - Unreleased - 2024-xx-xx

1.13.2 [4.0.1] - 2024-04-01

Changed

- Improved efficiency in packing

1.13.3 4.0.0 - 2023-12-04

Added

- Added new enumeration `Rotation` with values `Never`, `KeepFlat` and `BestFit`
- Added new `getAllowedRotation()` method to the `Item` interface to replace `getKeepFlat()`. This should return one of the new `Rotation` enum values
- Added new `generateVisualisationURL()` method to `PackedBox` and `PackedBoxList`. This will generate a custom URL for a visualisation you can access via the BoxPacker website
- Added new `packAllPermutations()` method to `Packer` to calculate all possible box combinations
- Added `throwOnUnpackableItem()` to `Packer` to control if an exception is thrown (or not) if an unpackable item is found (defaults to true, consistent with previous behaviour)
- Added `getUnpackedItems()` to `Packer` to retrieve the list of items that could not be packed (only applicable if exceptions are disabled)
- `PackedBox` now has readonly public properties `->box` and `->item`
- `PackedItem` now has readonly public properties `->item`, `->x`, `->y`, `->z`, `->width`, `->length`, `->depth`

Changed

- Minimum PHP version is now 8.2
- Exceptions are now in the `DVDoug\BoxPacker\Exception` namespace (previously `DVDoug\BoxPacker`)
- The signature of the `->canBePacked` method on the `ConstrainedPlacementItem` interface has been changed to replace the first two arguments(`Box $box`, `PackedItemList $alreadyPackedItems`) with `PackedBox $packedBox`. This allows callbacks to make use of the helper methods provided on `PackedBox`. Access to the box and items can be done via `$packedBox->box` and `$packedBox->items`
- `NoBoxesAvailableException` now has a `->getAffectedItems()` method instead of `->getItem()`. This should allow improved handling of the exception inside calling applications when multiple items cannot be packed

Removed

- Removed `getBox()` and `getItems()` from `PackedBox`. Use the new public properties instead
- Removed `->getItem()`, `->getX()`, `->getY()`, `->getZ()`, `->getWidth()`, `->getLength()` and `->getDepth()` from `PackedItem`. Use the new public properties instead
- Removed deprecated `ConstrainedItem`. You should use `ConstrainedPlacementItem` as a replacement
- Removed `getKeepFlat()` from the `Item` interface
- Removed `InfallliblePacker`. You can now get the same behaviour by calling `->throwOnUnpackableItem(false)` and `->getUnpackedItems()` on the main `Packer` class

1.13.4 3.12.1 - 2023-12-02

Fixed

- Restored ability to copy/paste the samples from the docs into a non-dev installation

1.13.5 3.12.0 - 2023-07-30

Changed

- Improved efficiency in packing

Removed

- Support for PHP 7.1, 7.2 and 7.3

1.13.6 3.11.0 - 2023-02-04

Changed

- Calling `json_encode()` on a `PackedBox` or `PackedItem` now additionally serialises the entire underlying `Box`/`Item` where those objects also implement `JsonSerializable`. Previously the serialisation only included the key values from the `Box`/`Item` interfaces themselves.

1.13.7 3.10.0 - 2022-09-10

Added

- Added `ItemSorter`, `BoxSorter` and `PackedBoxSorter` to allow calling applications to have better control over sorting decisions
- Added `beStrictAboutItemOrdering()` to `Packer` and `VolumePacker`

1.13.8 3.9.4 - 2021-10-21

Changed

- `psr/log` compatibility changed from `^1.0` to `^1.0||^2.0||^3.0`

1.13.9 3.9.3 - 2021-09-26

Fixed

- PHP8.1 deprecations

1.13.10 3.9.2 - 2021-07-04

Added

- Optional second parameter `$qty` to `ItemList->insert()`

Fixed

- Fixed issue where available width for an item could be miscalculated

Changed

- Improved memory usage

1.13.11 3.9.1 - 2021-05-05

Fixed

- Fixed issue where available width for an item could be miscalculated at the far end of a box

Changed

- Improved efficiency in packing in the vertical direction

1.13.12 3.9.0 - 2021-03-14

Added

- Added `packAcrossWidthOnly()` to `VolumePacker` for scenarios where the container will be side-loaded rather than top-loaded (e.g. truck loading)
- Added `getWeight()` helper method to `PackedItemList`
- Experimental visualisation tool has been added to the repo. All aspects of the tool are subject to change.

Changed

- Improved efficiency in packing

1.13.13 3.8.0 - 2021-01-26

Added

- Added `fromArray()` helper method to `BoxList` to make bulk add easier [bram123]

1.13.14 3.7.0 - 2021-01-01

Added

- Added `getVolume()` helper method to `PackedItemList`

1.13.15 3.6.2 - 2020-09-28

Added

- Support for PHP 8.0

1.13.16 3.6.1 - 2020-06-11

Fixed

- Fixed situation where internal `WorkingVolume` could be passed into a constraint callback, rather than the calling application's own `Box`
- Fixed issue where the list of previously packed items passed into a constraint callback was not correct

1.13.17 3.6.0 - 2020-04-26

Changed

- Improved efficiency in packing and weight distribution
- Major internal refactoring. The public-facing API did not change in any incompatible ways, but if you extended any of the `@internal` classes or made use of `@internal` methods you may be affected.
- Bail out earlier in the packing process where an item doesn't fit [colinmollenhour]

Fixed

- Fixed potential issue where internal sort consistency wasn't always correct
- Fixed potential issue where custom constraints might not be fully respected
- Avoid divide by zero error when a box is specified to have a depth of 0mm (e.g. 2D packing)
- Better docblocks [colinmollenhour]

1.13.18 3.5.2 - 2020-02-02

Changed

- Further optimisation when packing a large number of items

1.13.19 3.5.1 - 2020-01-30

Changed

- Optimisation when packing a large number of identical items

1.13.20 3.5.0 - 2020-01-26

Added

- Added a new interface `LimitedSupplyBox extends Box` for situations where there are restrictions on the number of a box type available for packing `Items` into. The interface contains 1 additional method `getQuantityAvailable()`.

- Added new exception `NoBoxesAvailableException` which is thrown when an item cannot be packed due to suitable boxes not being available (e.g. when the new functionality is used and the quantity available is insufficient). The existing `ItemTooLargeException` which is thrown when an item is too large to fit into any of the supplied box types at all (regardless of quantity) still exists, and now extends from `NoBoxesAvailableException` as a special case

Changed

- Improved efficiency in packing and weight distribution
- The `ItemList` passed to `VolumePacker`'s constructor is now cloned before usage, leaving the passed-in object unaffected. Previously this was used as a working dataset. The new behaviour aligns with the existing behaviour of `Packer`

Fixed

- Fixed issue where internal sort consistency wasn't always correct
- Some debug-level logging wasn't logging correctly

1.13.21 3.4.1 - 2019-12-21

Changed

- Speed improvements

1.13.22 3.4.0 - 2019-09-09

Added

- Added ability to specify that items are pre-sorted when creating an `ItemList` from an array

Changed

- Significant speed improvements when dealing with a large number of items

1.13.23 3.3.0 - 2019-07-14

Added

- Added `ConstrainedPlacementItem` as a more powerful version of `ConstrainedItem`

Changed

- Improved box selection for certain cases
- Speed improvements
- Increased detail in debug-level logging

Deprecated

- `ConstrainedItem` is now deprecated. Use `ConstrainedPlacementItem` instead

1.13.24 3.2.2 - 2018-11-20

Fixed

- Fixed divide by zero warning when attempting to pack an item with 0 depth

1.13.25 3.2.1 - 2018-11-13

Fixed

- Fixed issue where internal sort consistency wasn't always correct

1.13.26 3.2.0 - 2018-11-12

Added

- Added `getVolume()` helper method to `PackedItem` [Cosmologist]

Changed

- Improved item orientation selection for better packing
- Minor refactorings for code clarity

1.13.27 3.1.3 - 2018-06-15

Changed

- Worked around an PHP recursion issue when comparing 2 `Item`s.

1.13.28 3.1.2 - 2018-06-13

Changed

- Fixed typos in documentation code samples

1.13.29 3.1.1 - 2018-06-03

Changed

- Tweaked composer configuration to make it easier to run the samples in the documentation
- Minor speed improvements

1.13.30 3.1.0 - 2018-02-19

Added

- Optional 'Infallible' mode of packing to not throw an exception on error (e.g. item too large) but to continue packing the other items

Changed

- Improved stability algorithm
- Improved box selection for certain cases
- Some internal refactoring

1.13.31 3.0.1 - 2018-01-01

Added

- Declare `PackedBoxList` as implementing `Countable`

Changed

- Improved item orientation selection for better packing

1.13.32 3.0.0 - 2017-10-23

Added

- Introduced `PackedItem`s which are a wrapper around `Item`s with positional and dimensional information (x, y, z co-ordinates of corner closest to origin, width/length/depth as packed)
- Added method to set threshold at which weight redistribution is disabled

Changed

- `PackedBox` now contains a `PackedItemList` of `PackedItem`s (rather than an `ItemList` of `Item`s)
- `ConstrainedItem->canBePackedInBox()` now takes a `PackedItemList` of `PackedItem`s (rather than an `ItemList` of `Item`s)
- `BoxList`, `ItemList`, `PackedBoxList` have been altered to implement the `Traversable` interface rather than extend `SplHeap` directly so that any future changes to the internals will not need an API change
- Minimum PHP version is now 7.1

Removed

- HHVM support now that project has a stated goal of no longer targeting PHP7 compatibility

1.13.33 2.7.2 - 2020-09-28

Added

- Support for PHP 8.0

Removed

- Making the test suite compatible with PHP 8.0 has necessitated the removal of support for PHP 5.4 - 7.0 (see note below)

v2 of BoxPacker is in maintenance mode only, all users are encouraged to update to v3. This release has been made primarily to certify PHP 8 compatibility, unless an egregious bug is discovered (e.g. a physically impossible packing) this will probably be the last v2 release that includes any changes to core packing logic. (Any) further releases are intended to be limited to compatibility with future PHP versions.

1.13.34 2.7.1 - 2020-06-11

Fixed

- Fixed situation where internal `WorkingVolume` could be passed into a constraint callback, rather than the calling application's own `Box`
- Fixed issue where the list of previously packed items passed into a constraint callback was not correct

1.13.35 2.7.0 - 2020-04-26

Changed

- Improved efficiency in packing and weight distribution
- Major internal refactoring. The public-facing API did not change in any incompatible ways, but if you extended any of the `@internal` classes or made use of `@internal` methods you may be affected.
- Bail out earlier in the packing process where an item doesn't fit [colinmollenhour]

Fixed

- Fixed potential issue where custom constraints might not be fully respected
- Avoid divide by zero error when a box is specified to have a depth of 0mm (e.g. 2D packing)

1.13.36 2.6.5 - 2020-02-02

Changed

- Further optimisation when packing a large number of items

1.13.37 2.6.4 - 2020-01-30

Changed

- Optimisation when packing a large number of identical items

1.13.38 2.6.3 - 2020-01-26

Changed

- Improved efficiency in packing and weight distribution
- The `ItemList` passed to `VolumePacker`'s constructor is now cloned before usage, leaving the passed-in object unaffected. Previously this was used as a working dataset. The new behaviour aligns with the existing behaviour of `Packer`

Fixed

- Fixed issue where internal sort consistency wasn't always correct
- Some debug-level logging wasn't logging correctly

1.13.39 2.6.2 - 2019-12-21

Changed

- Speed enhancements

1.13.40 2.6.1 - 2019-09-15

Changed

- Speed enhancements

1.13.41 2.6.0 - 2019-07-14

Added

- Added `ConstrainedPlacementItem` as a more powerful version of `ConstrainedItem`

Changed

- Improved box selection for certain cases
- Speed improvements
- Increased detail in debug-level logging

Deprecated

- `ConstrainedItem` is now deprecated. Use `ConstrainedPlacementItem` instead

1.13.42 2.5.0 - 2018-11-20

Added

- Backported positional data support from v3 via new `getPackedItems()` method on `PackedBox`

Fixed

- Fixed divide by zero warning when attempting to pack an item with 0 depth

1.13.43 2.4.8 - 2018-11-13

Fixed

- Fixed issue where internal sort consistency wasn't always correct

1.13.44 2.4.7 - 2018-11-12

Changed

- Improved item orientation selection for better packing
- Minor refactorings for code clarity

1.13.45 2.4.6 - 2018-06-15

Changed

- Worked around an PHP recursion issue when comparing 2 `Items`.

1.13.46 2.4.5 - 2018-06-03

Changed

- Tweaked composer configuration to make it easier to run the samples in the documentation

1.13.47 2.4.4 - 2018-02-25

Changed

- Improved stability algorithm
- Improved box selection for certain cases
- Some internal refactoring

1.13.48 2.4.3 - 2018-01-01

Changed

- Improved item orientation selection for better packing

1.13.49 2.4.2 - 2017-10-23

Changed

- Previously 2 distinct item types could be mixed when sorting items for packing if they had identical physical dimensions. Now if all dimensions are identical, items are sorted by description so that they are kept together

1.13.50 2.4.1 - 2017-09-04

Fixed

- Used/remaining space calculations were sometimes offset by 90 degrees leading to confusing numbers

1.13.51 2.4.0 - 2017-08-14

Changed

- Significant reworking of core packing logic to clarify concepts used

Fixed

- Fixed issue where `getUsed[Width|Length|Depth]()` could sometimes return an incorrect value

1.13.52 2.3.2 - 2017-08-06

Changed

- In some cases, complex user-added constraints via `BoxPacker\ConstrainedItem` were not being obeyed
- Test classes refactored to be autoloadable
- Some internal refactoring

1.13.53 2.3.1 - 2017-04-15

Changed

- `PackedBox->getUsedDepth()` could incorrectly return a value of 0 in some situations

1.13.54 2.3.0 - 2017-04-09

Added

- Add callback system for more complex constraints e.g. max number of hazardous items in a box. To take advantage of the additional flexibility, implement BoxPackerConstrainedItem rather than BoxPackerItem

Changed

- Some internal refactoring

1.13.55 2.2.1 - 2017-03-12

Added

- Added `getItem()` to `ItemTooLargeException` to make it programmatically possible determine what the affected item is

1.13.56 2.2.0 - 2017-03-06

Added

- The previous limitation that all items were always packed flat has been removed
- A specific `ItemTooLargeException` exception is now thrown when an item cannot fit inside any boxes rather than a generic `\RuntimeException`

1.13.57 2.1.0 - 2017-01-07

Added

- Added `getUsed[Width|Length|Depth]()` on PackedBoxes to allow for better visibility into space utilisation

Changed

- Equal distribution of weight is now turned off when the number of boxes becomes large as it provides very little to no benefit at that scale and is slow to calculate
- Various optimisations and internal refactorings

1.13.58 2.0.2 - 2016-09-21

Changed

- Readme update to reflect v2 API changes

1.13.59 2.0.1 - 2016-09-20

Added

- Pass on the logger instance from the main Packer class into the helpers

Changed

- Allow unit tests to run with standalone PHPUnit

1.13.60 2.0 - 2016-05-30

There are no bugfixes or packing logic changes in v2.0 compared to the v1.5.3 release - the bump in version number is purely because the interface changed slightly.

Added

- Added a method to the Item interface to specify whether the item should be kept flat or not - this does not do anything yet, but adding now to avoid another major version bump later.

Changed

- Various refactorings to split out large functions into more readable pieces

Removed

- Removed `Packer->packIntoBox()`, `Packer->packBox()` and `Packer->redistributeWeight()`

1.13.61 1.7.3 - 2020-09-28

Added

- Support for PHP 8.0

Changed

- Optimisation when packing a large number of items
- Improved efficiency in packing and weight distribution
- The ItemList passed to VolumePacker's constructor is now cloned before usage, leaving the passed-in object unaffected. Previously this was used as a working dataset. The new behaviour aligns with the existing behaviour of Packer

Fixed

- Fixed issue where internal sort consistency wasn't always correct
- Fixed situation where internal WorkingVolume could be passed into a constraint callback, rather than the calling application's own Box
- Fixed issue where the list of previously packed items passed into a constraint callback was not correct

Removed

- Making the test suite compatible with PHP 8.0 has necessitated the removal of support for PHP 5.4 - 7.0 (see note below)

v1 of BoxPacker is in maintenance mode only, all users are encouraged to update to v3. This release has been made primarily to certify PHP 8 compatibility, unless an egregious bug is discovered (e.g. a physically impossible packing) this will be the last v1 release that includes any changes to core packing logic. (Any) further releases will be limited to compatibility with future PHP versions.

1.13.62 1.7.2 - 2019-12-21

Changed

- Speed enhancements

1.13.63 1.7.1 - 2019-09-15

Changed

- Speed enhancements

1.13.64 1.7.0 - 2019-07-14

Added

- Added `ConstrainedPlacementItem` as a more powerful version of `ConstrainedItem`

Changed

- Improved box selection for certain cases
- Speed improvements
- Increased detail in debug-level logging

Deprecated

- `ConstrainedItem` is now deprecated. Use `ConstrainedPlacementItem` instead

1.13.65 1.6.9 - 2018-11-20

Fixed

- Fixed divide by zero warning when attempting to pack an item with 0 depth

1.13.66 1.6.8 - 2018-11-13

Fixed

- Fixed issue where internal sort consistency wasn't always correct

1.13.67 1.6.7 - 2018-11-12

Changed

- Improved item orientation selection for better packing
- Minor refactorings for code clarity

1.13.68 1.6.6 - 2018-06-15

Changed

- Worked around an PHP recursion issue when comparing 2 `Items`.

1.13.69 1.6.5 - 2018-06-03

Changed

- Tweaked composer configuration to make it easier to run the samples in the documentation

1.13.70 1.6.4 - 2018-02-25

Changed

- Improved stability algorithm
- Improved box selection for certain cases
- Some internal refactoring

1.13.71 1.6.3 - 2018-01-01

Changed

- Improved item orientation selection for better packing

1.13.72 1.6.2 - 2017-10-23

Changed

- Previously 2 distinct item types could be mixed when sorting items for packing if they had identical physical dimensions. Now if all dimensions are identical, items are sorted by description so that they are kept together

1.13.73 1.6.1 - 2017-09-04

Fixed

- Used/remaining space calculations were sometimes offset by 90 degrees leading to confusing numbers

1.13.74 1.6.0 - 2017-08-27

API-compatible backport of 2.4.0. All features present except 3D packing.

Added

- Added `getUsed[Width|Length|Depth]()` on PackedBoxes to allow for better visibility into space utilisation
- Added callback system for more complex constraints e.g. max number of hazardous items in a box. To take advantage of the additional flexibility, implement BoxPackerConstrainedItem rather than BoxPackerItem
- A specific `ItemTooLargeException` exception is now thrown when an item cannot fit inside any boxes rather than a generic `\RuntimeException`
- Pass on the logger instance from the main Packer class into the helpers

Changed

- Significant reworking of core packing logic to clarify concepts used and split out large functions into more readable pieces
- Test classes refactored to be autoloadable and for unit tests to runnable with standalone PHPUnit
- Equal distribution of weight is now turned off when the number of boxes becomes large as it provides very little to no benefit at that scale and is slow to calculate

1.13.75 1.5.3 - 2016-05-30

Changed

- Some refactoring to ease future maintenance

1.13.76 1.5.2 - 2016-01-23

Changed

- Ensure consistency of packing between PHP 5.x and PHP7/HHVM

1.13.77 1.5.1 - 2016-01-03

Fixed

- Items were occasionally rotated to fit into space that was actually too small for them [IBBoard]

1.13.78 1.5 - 2015-10-13

Added

- Added method for retrieving the volume utilisation of a packed box

Changed

- Previously, when encountering an item that would not fit in the current box under evaluation, the algorithm would declare the box full and open a new one. Now it will continue to pack any remaining smaller items into the current box before moving on.

Fixed

- Boxes and items with large volumes were sometimes not sorted properly because of issues with integer overflow inside SplMinHeap. This could lead to suboptimal results. [IBBoard]

1.13.79 1.4.2 - 2014-11-19

Fixed

- In some cases, items that only fit in a single orientation were being recorded as fitting in the alternate, impossible one [TravisBernard]

1.13.80 1.4.1 - 2014-08-13

Fixed

- Fixed infinite loop that could occur in certain circumstances

1.13.81 1.4 - 2014-08-10

Changed

- Better stacking/depth calculations

1.13.82 1.3 - 2014-07-23

Fixed

- Fixed problem where available space calculation inadvertently missed off a dimension

1.13.83 1.2 - 2014-05-31

Added

- Expose remaining space information on a packed box

Fixed

- Fixed bug that preferred less-optimal solutions in some cases

1.13.84 1.1 - 2014-03-30

Added

- Support for HHVM

Changed

- Tweaked algorithm to allow limited lookahead when dealing with identical objects to better optimise placement
- Misc internal refactoring and optimisations

1.13.85 1.0.1 - 2014-01-23

Fixed

- Fixed issue with vertical depth calculation where last item in a box could be judged not to fit even though it would

1.13.86 1.0 - 2013-11-28

Added

- Generated solutions now have a second pass where multiple boxes are involved, in order to redistribute weight more evenly

Removed

- PHP 5.3 support

1.13.87 0.4 - 2013-08-11

Changed

- Minor calculation speedups

1.13.88 0.3 - 2013-08-10

Changed

- Now packs items side by side, not just on top of each other
- Generated solutions should now be reasonably optimal

1.13.89 0.2 - 2013-08-01

Added

- Supports solutions using multiple boxes

Changed

- API should be stable now - no plans to change it
- Generated solutions may not be optimal, but should be correct

1.13.90 0.1 - 2013-08-01

Initial release

Added

- Basic prototype
- Experimental code to get a feel for how calculations can best be implemented
- Only works if all items fit into a single box (so not production ready at all)