A new loading problem: product size reduction

François Schott,[1*] Sebastien Salmon,[1] Dominique Chamoret,[2]
Thomas Baron[3], Yann Meyer[4]

[1]My-OCCS, 2C Chemin de Palente, 25000 Besançon cedex, France

[2]ICB UMR 6303, CNRS Univ. Bourgogne Franche-Comté, UTBM, Belfort, France

[3]FEMTO-ST institute, univ. Bourgogne Franche-Comté, CNRS,

ENSMM Time and frequency dept. 26 Rue de l'épitaphe, 25030 Besançon cedex, France

[4]Univ. Bourgogne Franche-Comté, UTBM, Belfort, France[a]

[*]To whom correspondence should be addressed; E-mail: f.schott@my-occs.fr

[a]Research Associate at Sorbonne Universités, Université de Technologie de Compigne,

CNRS, UMR 7337 Roberval, Centre de recherche Royallieu, CS 60 319 Compigne cedex, France

Abstract

This paper introduces a new kind of cutting and packing problem (C.P.P.): the Product Size Reduction (P.S.R.). The cutting and packing problems are widely studied as NP-hard problems. The cutting and packing problems can be set by using different models. To solve those problems, optimization algorithms, such as conventional heuristics, meta-heuristics and tree search, are used. In addition, those algorithms may rely on positioning strategies. Usually, the loading problems are focused on logistics problems such as container or storage loading efficiency, whereas P.S.R. problems are related to electronics, transport and energy industries. The aim of this work is to solve a real case P.S.R. problem using a Particle Swarm Optimization (P.S.O.) algorithm. This case has two main features. First, all dimensions may vary between specific boundaries. Secondly, some objects have position constraints. A discrete space model has been built to simulate the objects loading. The optimization process is based on waterfalls objective-function (W.O.F.). Constraints are ordered and tested one after another. Depending on which constraints are fulfilled or not, a particular objective function is selected from a set. The P.S.O. algorithm manages to find a solution reducing significantly the volume.

Keywords - Cutting and packing problem/Product size reduction/Waterfalls objective function/Optimization/Particle Swarm Optimization

Table 1: Nomenclature

Notations	Unit	Description
$\mathbf{d}_{i,j}$	*mm*	Dimensions of object *i*, for axis *j*
$\mathbf{E}_{m,j}$	*mm*	Minimal position for all objects, for axis *j*
$\mathbf{E}_{M,j}$	*mm*	Maximal position for all objects, for axis *j*
f		objective function
f_k		*k*-th objective functions of the W.O.F. set
g_{best}		PSO algorithm particules global best result
i		objects increment, $i = 1, \cdots, p$
j		dimensions increment, $j = 1, \cdots, n$
k		Objective functions increment, $k = 1, \cdots, w$
n		Number of dimension, here $n = 3$
O_i		Object *i* to set
p		number of objects, here $p = 11$
p_{best}		PSO algorithm particule personal best result
r_i		Object *i* orientation
t		PSO algorithm current iterate
u_i		fixed orientation value for object *i*
$\mathbf{v}$		PSO algorithm particule velocity vector
V	*mm*	Explicit object volume
w		Size of W.O.F. set, here $w = 4$
W, D, H	*mm*	Explicit object dimensions, along *x*, *y*, *z* axises
$\mathbf{x}$		PSO algorithm particule position vector
$\mathbf{x}_{i,j}$	*mm*	Position of object *i*, for axis *j*
X, Y, Z	*mm*	Explicit object positions, along *x*, *y*, *z* axises

1 Introduction

In an industrial context, the effective space utilization is a real problem. It is very common subject in the logistic management. The main idea of C.P.P. is to set objects in space in order either to reduce or to maximize the used space ratio as shown in Figure 1. C.P.P. can be seen as geometric assignment problems : the goal is to distribute a set of cuboid objects in one cuboid containing space as an objective function is optimized and two basic geometric feasibility conditions hold (all objects lie entirely within the containing space and the objects do not overlap). From last decades, the number of publications about C.P.P. as widely increase. Nowadays, the most investigated case is the strip packing problem [1] consisting in loading a portion of a container. This paper discusses a new type of C.P.P., the Product Size Reduction (P.S.R.). The main idea of this problem is to set product components in order to reduce its size, with regards of design constraints. This point is very original compared to classical C.P.P.. Indeed, both the nature and the aim of this problem are different from others C.P.P.. This problem comes from the will of a company to reduce the size of an existing product, a rack. A new aspect is introduced in the problem: the design aspect. P.S.R. is related to electronics product, transport and energy industries.

C.P.P. problems are NP-hard and belong to the class of combinatorial optimization problems. NP-hard meaning that the time to solution increases exponentially as the size of the problem increases. They can be classified according to [2] classification. C.P.P. typology is a 3 steps one presented in Figure 2. The following criteria are required to fully classify a C.P.P..

Objective : Different objectives are possible. A problem may be to minimize a value, usually one dimension of the space, or to maximize it, as for instance, the space usage ratio.

Dimensions : Dimensions of the space(s) may be fixed, as for a cargo container, or variable, as for a piece of raw material, in which part will be cut. In variable dimensions case, one or several dimensions may vary.

Object heterogeneity : If all objects to set are the same, they are homogeneous, otherwise they are heterogeneous. The difference between strongly and weakly heterogeneous comes from the number of objects of a kind. If, in average, there are more than 6 objects of a kind, the problem is consider weekly heterogeneous [3].

Space heterogeneity : Some problems use several containing spaces to set objects, as for instance in the bin packing problem. In this case, spaces, like objects, may be homogeneous, strongly heterogeneous or weekly heterogeneous.

Number of dimensions : C.P.P. may be 1, 2 or 3 dimensions problems.

Additional constraints : Any constraints changing the nature of the problem.

If for a criteria, a problem does not belong to a case, it is a variant of a C.P.P.. If a problem has non orthogonal or non homogeneous material object or space, then it is also a variant of a C.P.P.. Objective, dimensions and object heterogeneity are required for the first step, detailed in Figure 3.

For each basic problem, many refined problems exist [2]. Only one example is stated for each problem. Though there are a C.P.P. typology. Refined problem names are not normed meaning that, in literature, a problem may have several name, as a name may refer to different problems.

There is 2 ways to set objects in space, either you do it directly as in [4], by setting coordinate, or un-directly, by using a positioning strategy as in [5]. Most of the time objects are set un-directly. Positioning strategies are meant to set object one after one. They often require the containing space to have an origin. Most of them have been design for 3D cuboid object and space. Positioning strategies include :

Layer and/or Wall : Objects are set so that they build walls and/or layers [3]. A width is usually defined based on the first set object. Objects can't be set if they outreach this width. Walls and layer often relies on stacks.

Stack : Objects are put one after one on each others top in order to build stack [3]. It may be use as an improvement to set several object at a time.

Reference point : The principle is that the solving process will try to set object at a list of possible positions given by reference point. When an object is set, the reference point where it belongs is replaced by the limits of the object along each axis, with the previous point as a reference. With 3D cuboid object, it is like replacing the origin corner of an object by the 3 corners along the x,y,z axis [6].

Extreme points : It is similar to reference points. In the case of 3D cuboid objects, the original point will be replaced by all the other corners [7].

Maximal space : This strategy considers free space to position objects. Objects will be set at the origin corner of the biggest free space. Everytime an object is set, up to *n* new free space will be created from the former space. For each axis, A new space is created by cutting the occupied portion of the former space [8].

Cage bin : Cage bin is the smallest virtual object containing all set objects. the gaol of this strategy is to keep it as small as possible [6].

D.B.L.F. : Deepest-bottom-lef fitt (D.B.L.F.) is setting objects at the deepest, then closest to the bottom, then leftest point possible [9]. Usually it works with reference points. It is like sorting free position by alphabetic order and trying every object, one after one, until one can fit. If no object can fit in, the position is removed from the list of free position.

Anchor distance : It is similar to D.B.L.F. exept that free position will be sorted by their normed distance to the closest corner of the space [8]. As one may notice it works only with fixed dimensions space.

Best fit : Objects are set in a way that the area in contact with others objects is maximized [10].

First fit : Objects are set at the first possible position. Due to its simplicity, this strategy is commonly used [11].

Some techniques may be used to improve the process.

Sorting : Object are sorted, ussually from the higher volume to the lower one, in order to be set in a more efficient way [12]. It is the most common improvement done.

Block building : Assemble objects into block to set several object at a time [7].

Moving along axis : after an object being set, a subroutine will try to move it along each axis, in the origin direction [6].

Inflating : After an object is set, a subroutine will try to increase its dimensions on every axis at the same time until a collision occured. after doing so the subroutine will try to replace the original object by a new bigger one that can be included in the inflated object [13].

Shuffeling : From time to time a subroutine will remove objects from a space and replace them in a new random order [13].

Shacking : from time to time, objects will be moved from one corner of the space to the opposite one and back [14].

Reference length : Objects cannot be set if they overreach the reference length along an axis. When no object can be set, this reference length is increased [7].

Another feature of those problems is the requirement to develop powerful numerical tools to handle the wide variety and complexity of shapes that need to be packed. A C.P.P. problem is a complex optimization geometric modeling in high-dimensional space with nonconvex and disconnected space of possible solutions. In literature, four different ways to numerically model C.P.P. have been found. Each of them have properties, advantages and drawbacks that can be found in Table 2.

M.I.L.P. (mixed-integer linear programming): It consists of using a mathematical model of the problem, based on a set of equation and in-equation [15]. It requires a good understanding of the problem. The harder the problem the harder it will be to implement the model.

Cuboid shape : Consists of simulating cuboid shape loading in order to solve the problem [16]. Objects have properties, such as dimensions, position or extremity, that are set, modified and used for computation. This modeling cannot solve exactly problems with complex shape which is not a big issue as C.P.P. are focused on logistics where most of the application rely on cuboid shape.

Discrete space : In this modeling objects and space are composed of discrete elements [17]. Usually, unit length cubes are chosen. As in cuboid shape objects loading is simulated. Discrete space may require substantial computational resources. This model is a good compromise between solution quality and efficiency for problems involving complex shapes.

Complex shape : Consists of modeling objects the same way a CAD software does it [12]. This kind of modeling requires either to use a CAD software or to recreate some of its features. Complex shape is the only way to solve exactly C.P.P. with complex shapes. Unfortunately, it is hard to implement, explaining why its almost never used.

As specified before, A C.P.P. problem is a complex optimization problem. Due to its complexity, three type of approaches are usually used to solve it [18].

Conventional heuristics : Heuristic algorithm [19] are defined as a methods specific to a problem to find a satisfying solution in a short amount of time. This definition is large and can include many algorithm such as constructive cooperative coevolution, memetic, gaussian adaptation or GRASP algorithm.

Metaheuristics : Meta-heuristic algorithm are diferent from heuristic so they are meant to be generic and not to solve a particular type of problem. They include many well-known optimization algorithm famillies such as Genetic algorithms [20], Ant Colony Optimization [21, 22], Particle Swarm Optimization [23], Artificial Bee Colony (ABC) [24]

Tree-search-based methods [3] : Those algorithm are used in the case where problem could be solved one step at a time. At each steps the algorithm consider every solution as a branch of a tree. After testing all solution the best one is chose and the algorithm goes to the next step.

2 Product Size Reduction

This section introduce the P.S.R. problem to be solved. First, problem characteristics will be discuss (2.1). Then, object position and problem constraints will be detailed in sections 2.2 and 2.3. Finally, Problem will be stated (2.4).

2.1 Overview

According to C.P.P. typology, P.S.R., presented in Figure 4, is a 3D open dimension problem. Indeed, the goal is to minimize a value, the volume occupied by objects. At least one dimension is not fixed, all in this case. The objects are strongly heterogeneous, though it could have been otherwise. It is a 3D case. This problem is special instance of P.S.R. as it has additional constraints, the one about objects position. Actually, this P.S.R. has two interesting features. First, all dimensions may vary under limits, meaning no dimension of the product can be increased. Second, some components have position constraints. This will be detailed in section 2.2.

The problem presented in this paper is a real life industrial problem, where a company wants to reduce the size of an already existing product. In this P.S.R.

problem, 11 electronic components have to be set in a rack. Details of components can be found in Table 3.

2.2 Objects position

In this case, the objects are oriented along axis. So, every object position is given by three translations and one orientation [6]. Indeed, for a cuboid form if directions along an axies are not taken into account, there are only six possible orientations instead of twenty four [17]. Possible orientations can be found in Figure 5. For every translation, there are 500 possible values, corresponding to the 500 elements side discrete working space detailed in 3.2. As a reminder from 3, there are 11 objects of 9 kinds. Every object has 3 translations (500 possible values each) and 1 orientation (6 possible values). By an easy computation, it can be deduced that there are more than 8 billions possible solutions to this problem. A small amount of them will satisfy the constraints.

2.3 Constraints

This problem main constraint is that objects cannot overlap. On the opposite, full support [16], guillotine cut [25] or weight repartition [26] constraints are not applied to this problem. Full support means that objects have to stand, completely, on the bottom of the space or on top of another object. As mentioned in Section 1, some objects have position and/or orientation constraints [27]. For instance, some components should be set in the front of the rack, meaning their x coordinate should be the lowest of all components. As objects are position according to the space and not to other objects they have absolute position. According to [27], this constraints applies to only 2.5% of literature.

2.4 P.S.R. statement

P.S.R. problem goal is to minimize the volume occupied by all objects. In This approach, volume evaluation is based on the volume of the smallest cuboid containing all the objects. To this end, the minimal (resp. maximal) position of all the objects is evaluated for each axis as follows :

$$\forall j = 1, \cdots, n \qquad \mathbf{E}_{m,j} = \min_{i=1,\cdots,p} \{\mathbf{x}_{i,j}\} \tag{1}$$

$$\forall j = 1, \cdots, n \qquad \mathbf{E}_{M,j} = \max_{i=1,\cdots,p} \{\mathbf{x}_{i,j} + \mathbf{d}_{i,j}\} \tag{2}$$

Then the volume is defined as a simple product of $n = 3$ lengths :

$$f(\mathbf{E}_{m,j}, \mathbf{E}_{M,j}) = \prod_{j=1}^{n} \left(\mathbf{E}_{M,j} - \mathbf{E}_{m,j}\right) \tag{3}$$

As mentioned in section 2.3, P.S.R. is constrained. Two types of constraints are defined. Equations (4) and (5) are respectively for position and orientation constraints. Equation (4) states that for any object, for every axis, it is not allowed that lowest object lower limit is inferior to space lower limit or that highest object upper limit is higher than space upper limit. Equation (5) states that there is a list of dual values defining that a given object's orientation must be equal to a fixed value.

$$(C_1) \quad \forall(a,b), \quad (\mathbf{x}_{a.b} = \mathbf{E}_{m.b}) \vee (\mathbf{x}_{a.b} + \mathbf{d}_{a.b} = \mathbf{E}_{M.b}) \tag{4}$$

$$(C_2) \quad \forall(k,u_i), \quad r_k = u_i \tag{5}$$

Equation (6) states that objects should not overlap. It states that for any couple of objects, it is not allowed that, for all axis, first object lower limit is inferior to second object lower limit and first object upper limit is superior to the second object lower limit.

$$(C_3) \quad \forall(i1,i2), \big(\forall j, ! \left(\mathbf{x}_{i1.j} + \mathbf{d}_{i1.j} > \mathbf{x}_{i2.j}\right) \wedge \left(\mathbf{x}_{i1.j} < \mathbf{x}_{i2.j}\right)\big) \tag{6}$$

Finally, the P.S.R. problem can be formulated as a classical optimization problem under constraints:

$$\begin{cases} \min f(\mathbf{E}_{m.j}, \mathbf{E}_{M.j}) \\ \\ + \quad (C_1), (C_2), (C_3) \end{cases} \tag{7}$$

3 Solving Approach

This section present the approach used to solve the P.S.R. problem. First, the overall approach will be presented (3.1). Second, the chosen numeric model will be disused (3.2). Third, the model functioning will be detailed (3.3).

3.1 Global Approach

A direct positioning approach has been chosen. Indeed, even if un-direct positioning is more efficient, it does not fit well with the position constraints. Un-direct positioning relies on positioning strategies, wich tends to set objects on deepest-bottomest-leftest position available. Though, it may be tough, or impossible, using those strategies, to set objects so they are in one face of the product. In this section, an overview of the whole solving process is presented. It can be seen in Figure 6.

The heart of the solving approach is the loop between the optimization software and the numerical simulation, which is the discrete space representation of the P.S.R. problem. Optimization process searches the best positions for objects thanks to an algorithm trying configuration. Every time the optimization software tries a configuration, it sends it to the simulation, which is giving back the results. The optimization software solves the optimization problem given by equation (7). It requires classical data : design variables, objective function, constraints and algorithm settings. For this optimization case, the design variables are objects translations and orientations. The simulation also needs some information, like spaces dimensions, objects dimensions or objects locked positions. Indeed, it has been chosen to replace, inside the simulation, some variables values by others to fix some objects in space. Fixing some objects along axis is the easiest way to solve position constraints without having objects to diverge in space. Models information are set directly inside the model. Optimization case and model information rely on some preliminary computations, such as defining the side length of an element of the discrete space. Those computations depend on the problem and should be done again every time a new P.S.R. problem has to be solved. Finally, when the loop between Optimization software and simulation has found a result, result can be exploited. The result should be implemented, using a CAD software, here freeCAD [28] ($https://www.freecadweb.org/$), in order to validate and fully exploit it.

3.2 Discrete Space

To avoid confusion the difference between space, dimensions and volume, should be stated. A space is a n dimensions area, in which objects are set. As a finite dimensions space is used, for each dimension, the space as a beginning and an end. The product of those lengths is the volume (Equation 3).

A discrete space model has been used to solve this problem. This choice has been made to be able to take into account eventual complex shape. The space is made of $5mm$ side length cube. The overall space is a 500 elements side cube. Those sizes has been chosen to match system memory capacities.

To solve this problem, 4 spaces has been defined.

- The overall space is the simulation discrete space.
- The working space is the space where objects should be set to have a chance

to solve the problem.

- The current space is the size of the existing rack.
- The target space is the goal to reach.

All spaces origins are at the deepest-bottom-left corner. The work space position, relative to the overall space, is so that their centers share the same position. Position of Current space and target space depends on objects position in overall space.

3.3 Simulation

As described in Section 3.2, the solving process relies on a numerical simulation. An explanation flowchart of this simulation can be found in Figure 7.

In order to converge to the optimal solution, a new type of objective function has been implemented: waterfalls objective-function. W.O.F. is a way to take constraints into account, close to constraints relaxation [29] and constraints ordering methods [30]. Constraints relaxation uses constraints to reduce space search. Constraint ordering methods are so that problem is solved by a decision tree where constraints are added one by one at each step. In W.O.F. case, constraints are ordered and tested one after another. Depending on which constraints are fulfilled or not, a particular objective-function is selected from a set. The flowchart 8 presents this case W.O.F.. Note that it is very important that objective-functions outputs intervals do not overlap. This is the hardest part of the W.O.F. implementation.

4 Optimization Algorithm

This section highlight the P.S.O.-starcraft algorithm used to solved this problem [31]. In a first time, regular P.S.O. features will be presented (4.1). In a second time, P.S.O.-starcraft particularity will be detailed (4.2).

4.1 P.S.O.

To solve problem (7), a P.S.O. algorithm (Particle Swarm Optimization) has been used. It is a global optimization algorithm described as sociologically inspired. The Particle Swarm Optimization (P.S.O.) algorithm belongs to the category of swarm intelligence techniques. In P.S.O., each solution of the optimization problem is regarded as a particle in the search space, which adjusts its position in the search space according to its own flying experience and the flying experience of other particles [32]. The P.S.O. algorithm has only a small number of parameters which need to be adjusted and is easy to implement.

In a basic P.S.O. algorithm, members of a swarm fly in the search field (of n dimensions) and each member is attracted by its personal best solution and by the best solution of its neighbours [33]. Each particle has a memory storing all data relating to its flight (location, speed and its personal best solution). It can also inform its neighbors, i.e. communicate its speed and position. This ability is known as socialization. For each iteration, the objective function is evaluated for every member of the swarm. Then the leader of the whole swarm can be determined: it is the particle with the best personal solution. The process leads at the end to the best global solution. At each iteration t , the location and speed of one particle are updated as follows:

$$\begin{cases} \mathbf{V}_{t+1} = \omega_t \mathbf{V}_t + c_1 r_1 (pbest - \mathbf{x}_t) + c_2 r_2 (gbest - \mathbf{x}_t) \\ \mathbf{x}_{t+1} = \mathbf{V}_{t+1} + \mathbf{x}_t \end{cases} \tag{8}$$

pbest is the personal best previous position of the particle and *gbest* is the best global position among all the particles in the swarm (Figure 9). The parameters r_1 and r_2 are two random numbers between 0 and 1. The constant c_1 and c_2 represent trust parameters indicating how much confidence the current particle has in itself and how much confidence it has in the swarm. Theses acceleration constants c_1 and c_2 indicate the stochastic acceleration terms which pull each particle toward the best position attained by the particle or the best position attained by the swarm. The role of the inertia weight ω is considered important for the convergence behaviour of P.S.O. algorithm. The inertia weight is employed to control the impact of the previous history of velocities on the current velocity. Thus, the parameter ω regulates the trade off between the global (wide ranging) and the local (nearby) exploration abilities of the swarm. A proper value for the inertia weight provides balance between the global and local exploration ability of the swarm, and thus results in better solutions. Numerical tests imply that it is preferable to initially set the inertia to a large value, to promote global exploration of the search space, and gradually decrease it to obtain refined solutions. Weight of particles is decreasing through time in order to fit the search area shrinking. The weight is decreasing according to the equation (9) law:

$$\begin{cases} w_t = w_{max} - \Delta.(t)/(it_{max}) \\ \Delta = w_{max} - w_{min} \end{cases} \tag{9}$$

An improved version of P.S.O. is used: BSG-Starcraft Radius improvements [31].

4.2 P.S.O.-starcraft

P.S.O.-starcraft has two improvements compared to the original one. First, the radius improvements. A swarm radius is computed using an infinite norm. If this radius is inferior to a certain value the algorithm stop as it considers it has found a global solution. Radius notion is displayed in Figure 10.

Second, a starcraft improvement. At each iteration, there is a probability that the leader sends a group of new random fast particles called raptors as shown in 11.

If one raptors find a better solution than the leader, the whole swarm will move to this position, conserving its geometry with respect to the leader. A pseudo code of the P.S.O.-starcraft algorithm can be found in algorithm 1.

Algorithm 1 P.S.O. starcraft pseudo code

```
Require: Initialization
 1: Initialize $n$ particles : random position ($x$), weight ($w$) random velocity ($v$)
 2: Evaluate particules ($f$)
 3: Find particles personal best ($p_{best}$) and global best ($g_{best}$)
 4: compute swarm radius ($r$)
 5: while $t \leq t_{max}$ and $r < threshold$ do
 6:     $g_{best}$ become the carier
 7:     if $random > startcraft - threshold(s_t)$ then
 8:         create $n_r$ random raptors : random long range displacement from the carier
 9:         Evaluate raptor ($f_{raptors}$)
10:         find the best raptor ($r_{best}$)
11:         if $f(r_{best}) < f(g_{best})$ then
12:             jump the swarm to the best raptor position
13:         end if
14:     end if
15:     evaluate particules ($f$)
16:     update personal best ($p_{best}$), global best ($g_{best}$), velocity ($v$), weights ($w$) and
        position ($x$)
17:     compute swarm radius ($r$)
18: end while
```

Table 4 gives settings used values for the P.S.O.-starcraft algorithm. If a value is between braces it means it has no impact on algorithm behaviour in this case.

5 Results and discussion

This section discuss result obtained thanks to the methods presented in previous sections. Results presented in section 5.1 have been implemented as in section 5.2. Some issues have been faced (5.3).

5.1 Result

Table 5 highlights the product size reduction. At the end, the rack volume has been reduced by 12%. The client is mostly interested by reducing the width, which has been reduced by 8%.

Figure 12 displays the components organization in space before and after optimization. Detail of components position is summed up in table 6. The 2 main differences are, first the biggest components has moved from the left side to the right one and second, similar components are packed.

5.2 Implementation

The problem has been solved using a computer with the specifications presented in table 7. My-OCCS optimization software [34] ($http : //my-occs.fr/software.htm$) has been used to solved this problem. To solve this problem, 7239 iterations were required, which last about 3 hours with the defined computer (1.6 seconds per iteration in average). So, using only one core, model execution, takes about 400 ms.

5.3 Issues

Many issues have been faced. At the beginning objects tend to move freely into overall space until some of them try to go outside leading to computer crash. To fix this, some variables values, corresponding to translation, are replaced by fixed a values, so that for every axis at least one object is fixed. Object with position constraints are fixed in priority. Most of the issues faced were related to memory limitations and model efficiency requirements to have an accepatble computational time. To solve those issues, the following elements have been used : reduce overall and working spaces sizes, increase discrete space unit element side length, collect information along the loading process to make collision computation easier, improve collision function. The hardest issues faced was the way to take into account constraints into the objective function computation. It has been faced by using W.O.F. has mentioned in section 3.3.

6 Conclusion

This paper has introduced a new kind of cutting and packing problem: Product Size Reduction. This problem comes from the will of a company to reduce the size of its existing rack.

P.S.R. is a 3 dimensions open dimension problem with 2 interesting features, all dimensions may vary under limits and some components have position constraints. This P.S.R. has 11 objects. Every object position is given by 3 translation and 1 orientation among 6. This leads to more than 8 billion possible solutions. Collision constraints are considered. The goal is to minimize the volume occupied by all objects and how to check constraints.

A direct positioning approach has been chosen. The heart of this approach is the loop between optimization software and simulation. To settle this approach , the optimization software requires optimization case and simulation needs information. Optimization case and model information rely on some preliminary computations. The result, obtained by this approach, should be implemented in a CAD software for validation and exploitation. The P.S.R. has been solved by the P.S.O.-starcraft algorithm. I

The following results have been achieved. The rack width has been reduce by 8% and the volume by 12%. Similar components have been packed.

Acknowledgement

This work was partly supported by the FEDER project SMART-INN under grant FC0001257 - 11002 . This work was partly supported by the industrial Group Galilé ($http : //www.galile.fr/$) under Galilé 360 grant. The author would like to acknowledge Peinta ($http : //www.peinta.com/$) for the industrial application presented in this paper.

References and Notes

[1] Jianli Chen, Wenxing Zhu, and Zheng Peng. A heuristic algorithm for the strip packing problem. *Journal of Heuristics*, 18(4):677–697, 2012.

[2] Gerhard Wascher, Heike Haussner, and Holger Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130, December 2007.

[3] Sheng Liu, Wei Tan, Zhiyuan Xu, and Xiwei Liu. A tree search algorithm for the container loading problem. *Computers & Industrial Engineering*, 75:20–30, September 2014.

[4] Mhand Hifi, Imed Kacem, Stephane Negre, and Lei Wu. A linear programming approach for the three-dimensional bin-packing problem. *Electronic Notes in Discrete Mathematics*, 36:993–1000, 2010.

[5] Andreas Bortfeldt and Daniel Mack. A heuristic for the three-dimensional strip packing problem. *European Journal of Operational Research*, 183(3):1267–1279, December 2007.

[6] Yaohua He, Yong Wu, and Robert de Souza. A global search framework for practical three-dimensional packing with variable carton orientations. *Computers & Operations Research*, 39(10):2395–2414, October 2012.

[7] Lijun Wei, Wee-Chong Oon, Wenbin Zhu, and Andrew Lim. A reference length approach for the 3d strip packing problem. *European Journal of Operational Research*, 220(1):37–47, July 2012.

[8] Wenbin Zhu and Andrew Lim. A new iterative-doubling greedy-lookahead algorithm for the single container loading problem. *European Journal of Operational Research*, 222(3):408–417, November 2012.

[9] Kyungdaw Kang, Ilkyeong Moon, and Hongfeng Wang. A hybrid genetic algorithm with a new packing strategy for the three-dimensional bin packing problem. *Applied Mathematics and Computation*, 219(3):1287–1299, October 2012.

[10] S. D. Allen, E. K. Burke, and G. Kendall. A hybrid placement strategy for the three-dimensional strip packing problem. *European Journal of Operational Research*, 209(3):219–227, March 2011.

[11] F. K. Miyazawa and Y. Wakabayashi. Three-dimensional packings with rotations. *Computers & Operations Research*, 36(10):2801–2815, October 2009.

[12] Mikhail Verkhoturov, Alexander Petunin, Galina Verkhoturova, Konstantin Danilov, and Dmitry Kurennov. The 3d object packing problem into a parallelepiped container based on discrete-logical representation. *IFAC-PapersOnLine*, 49(12):1–5, 2016.

[13] Wenbin Zhu, Zhaoyi Zhang, Wee-Chong Oon, and Andrew Lim. Space defragmentation for packing problems. *European Journal of Operational Research*, 222(3):452–463, November 2012.

[14] Tony Wauters, Jannes Verstichel, and Greet Vanden Berghe. An effective shaking procedure for 2d and 3d strip packing problems. *Computers & Operations Research*, 40(11):2662–2669, November 2013.

[15] Yao-Huei Huang, F. J. Hwang, and Hao-Chun Lu. An effective placement method for the single container loading problem. *Computers & Industrial Engineering*, 97:212–221, July 2016.

[16] Yanira Gonzalez, Gara Miranda, and Coromoto Leon. Multi-objective multi-level filling evolutionary algorithm for the 3d cutting stock problem. *Procedia Computer Science*, 96:355–364, 2016.

[17] Youn-Kyoung Joung and Sang Do Noh. Intelligent 3d packing using a grouping algorithm for automotive container engineering. *Journal of Computational Design and Engineering*, 1(2):140–151, April 2014.

[18] Tobias Fanslau and Andreas Bortfeldt. A Tree Search Algorithm for Solving the Container Loading Problem. *INFORMS Journal on Computing*, 22(2):222–235, May 2010.

[19] Lijun Wei, Qian Hu, Stephen C. H. Leung, and Ning Zhang. An improved skyline based heuristic for the 2d strip packing problem and its efficient implementation. *Computers & Operations Research*, 80:113–127, April 2017.

[20] Jia-Nian Zheng, Chen-Fu Chien, and Mitsuo Gen. Multi-objective multi-population biased random-key genetic algorithm for the 3-D container loading problem. *Computers & Industrial Engineering*, 89:80–87, November 2015.

[21] Juan Rada-Vilela, Manuel Chica, Oscar Cordon, and Sergio Damas. A comparative study of Multi-Objective Ant Colony Optimization algorithms for the Time and Space Assembly Line Balancing Problem. *Applied Soft Computing*, 13(11):4370–4382, November 2013.

[22] Yanxin Xu, Gen Ke Yang, Jie Bai, and Changchun Pan. A Review of the Application of Swarm Intelligence Algorithms to 2d Cutting and Packing Problem. In Ying Tan, Yuhui Shi, Yi Chai, and Guoyin Wang, editors, *Advances in Swarm Intelligence*, volume 6728, pages 64–70. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. DOI: 10.1007/978-3-642-21515-5_8.

[23] Takwa Tlili and Saoussen Krichen. On solving the double loading problem using a modified particle swarm optimization. *Theoretical Computer Science*, 598:118–128, September 2015.

[24] Suman Samanta and Shankar Chakraborty. Parametric optimization of some non-traditional machining processes using artificial bee colony algorithm. *Engineering Applications of Artificial Intelligence*, 24(6):946–957, September 2011.

[25] Jose Fernando Goncalves and Mauricio G. C. Resende. A biased random key genetic algorithm for 2d and 3d bin packing problems. *International Journal of Production Economics*, 145(2):500–510, October 2013.

[26] H. Gehring and A. ortfeldt. A genetic algorithm for solving the container loading problem. *International Transactions in Operational Research*, 4(5):401–418, 1997.

[27] Andreas Bortfeldt and Gerhard Wascher. Constraints in container loading, a state-of-the-art review. *European Journal of Operational Research*, 229(1):1–20, 2013.

[28] Yorik van Havre Juergen Riegel, Werner Mayer, 2016.

[29] Christian Bessiere. Chapter 3 - constraint propagation. In Peter van Beek Francesca Rossi and Toby Walsh, editors, *Handbook of Constraint Programming*, volume Volume 2, pages 29–83. Elsevier, 2006.

[30] Miguel A. Salido. A non-binary constraint ordering heuristic for constraint satisfaction problems. *Applied Mathematics and Computation*, 198(1):280–295, April 2008.

[31] Dominique Chamoret, Sébastien Salmon, Noelie Di Cesare, and Yingjie J. Xu. *BSG-Starcraft Radius Improvements of Particle Swarm Optimization Algorithm: An Application to Ceramic Matrix Composites*, pages 166–174. Springer International Publishing, Cham, 2014.

[32] J. Kennedy and R.C. Eberhart. Particle swarm optimisation. In *Proceedings of the IEEE International Conference on Neural Networks*, 1995.

[33] F. van den Bergh and A.P. Engelbrecht. A study of particle swarm optimization particle trajectories. *Information Sciences*, 176(8):937 – 971, 2006.

[34] Sebastien Salmon and Francois Schott, 2015.

List of Tables

List of Figures

Table 2: Model Characteristics

	M.I.L.P.	Cuboid shape	Discrete space	Complex shape
Nature	Math. Model	Simu.	Simu.	Simu. - CAD
Complex object	No	No	Approx.	Yes
Model efficiency	+++++	++++	++	+
Implementation easiness	++	+++++	++++	+
Popularity	++++	+++++	++	+

Table 3: Objects details

Object	NB	W	D	H	Position	Orientation
Object 1	1	400	210	60	Bottom	
Object 2	1	335	290	240		
Object 3	1	120	240	260	Front	Depth
Object 4	1	80	40	110	Front	Depth
Object 5	1	100	80	45	Top — bottom	Heigth
Object 6	1	50	100	30	Front	Depth
Object 7	2	120	200	40		
Object 8	2	260	30	60		
Object 9	1	60	100	90		

Table 4: P.S.O. starcraft settings

Parameter	Value	Remarks
d	44	Number of design variables
It_{max}	10000	Iteration limit
n	100	Number of particles of the swarm
c_1	1	Personnal best factor
c_2	1	Global best factor
w_{min}	0.7	Minimal inertia factor
w_{max}	0.9	Maximal inertia factor
$Radius$	0	Activation of radius 1 else 0
r_t	(10)	Number of iteration inside radius to stop
$Starcraft$	0	Activation of starcraft 1 else 0
s_t	(0.9)	Starcraft activation probability
r_a	(2)	Raptor jump factor
n_r	(10)	Number of raptors

Table 5: Result

Rack	W	D	H	V
Initial	510	525	260	69.62
Final	470	500	260	61.10

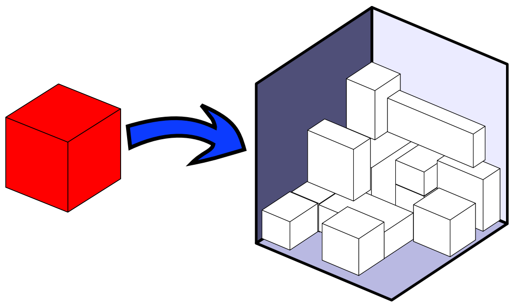


Figure 1: C.P.P. main idea

Table 6: Components position

Components	Color	X	Y	Z	W	D	H
Object 1	Purple	0	385	0	400	60	210
Object 2	Grey	0	50	0	290	335	240
Object 3	Orange	350	0	0	120	240	260
Object 4	Yellow	290	0	175	40	110	80
Object 5	Brown	190	0	0	80	45	100
Object 6	Blue	300	0	30	50	100	30
Object 7.1	Green	310	280	5	120	40	200
Object 7.2	Green	310	320	5	120	40	200
Object 8.1	Red	440	240	0	20	260	60
Object 8.2	Red	440	240	60	20	260	60
Object 9	Black	290	180	25	60	90	100

Table 7: Computeur specifications

Component	specification
Computeur	ASUS DESKTOP-B140V12 Z10PE-D16 WS
Motherboard	Asus Z10PE-D16WS
Processor	2 x (Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz, 6 cores, 12 Threads)
RAM	64 GB - 8 x (Micron crucial Premium memory 8GB DDR4-2133)
Graphic Card	Saphire R250 (R7)
OS	Windows 10 Professionnal

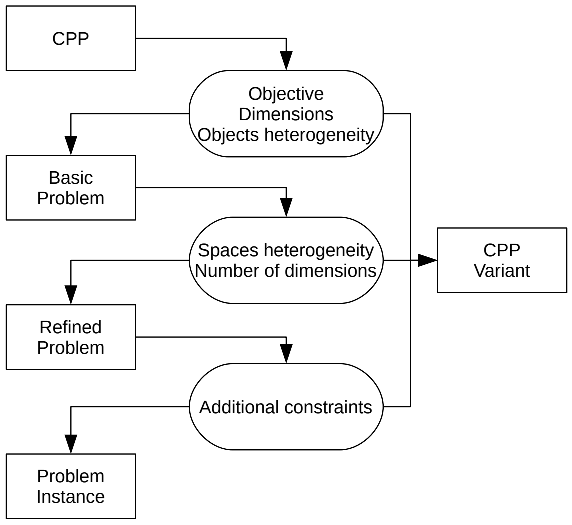


Figure 2: C.P.P. Typology steps

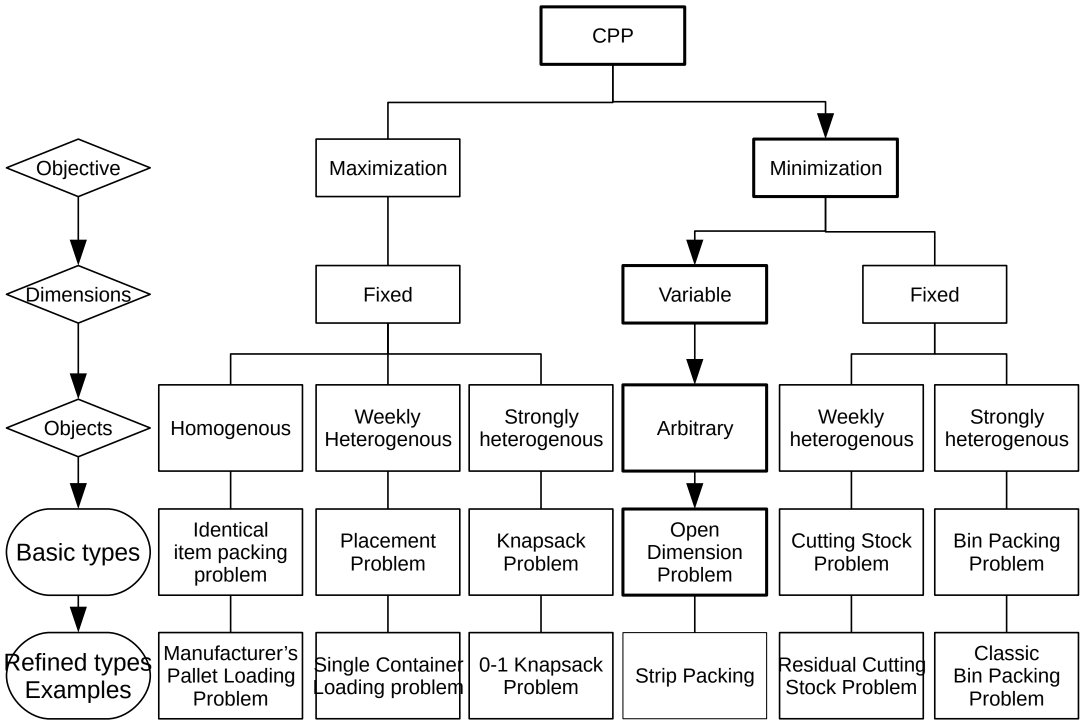


Figure 3: C.P.P. Typology

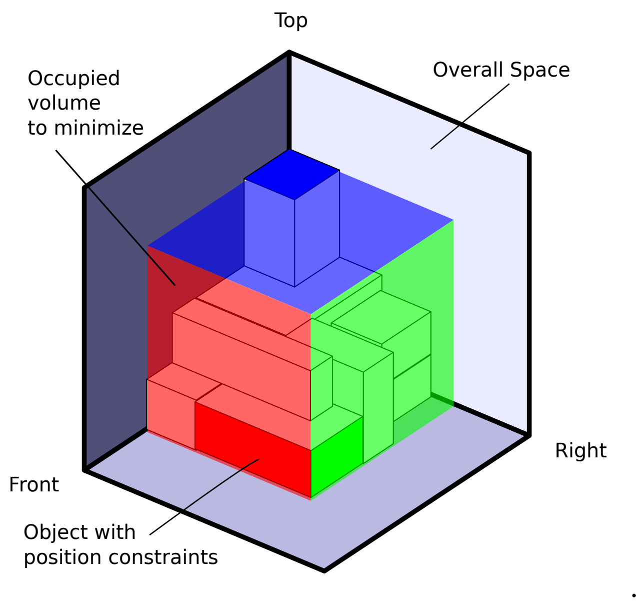


Figure 4: P.S.R. main idea

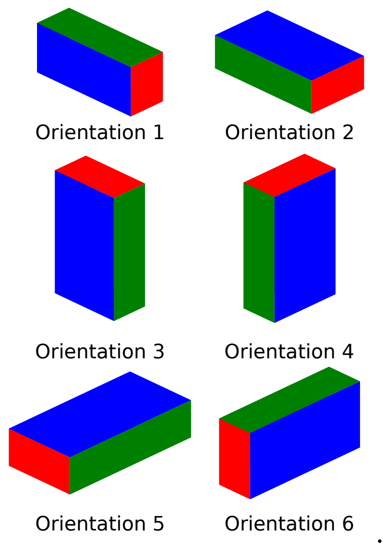


Figure 5: Objects Orientations

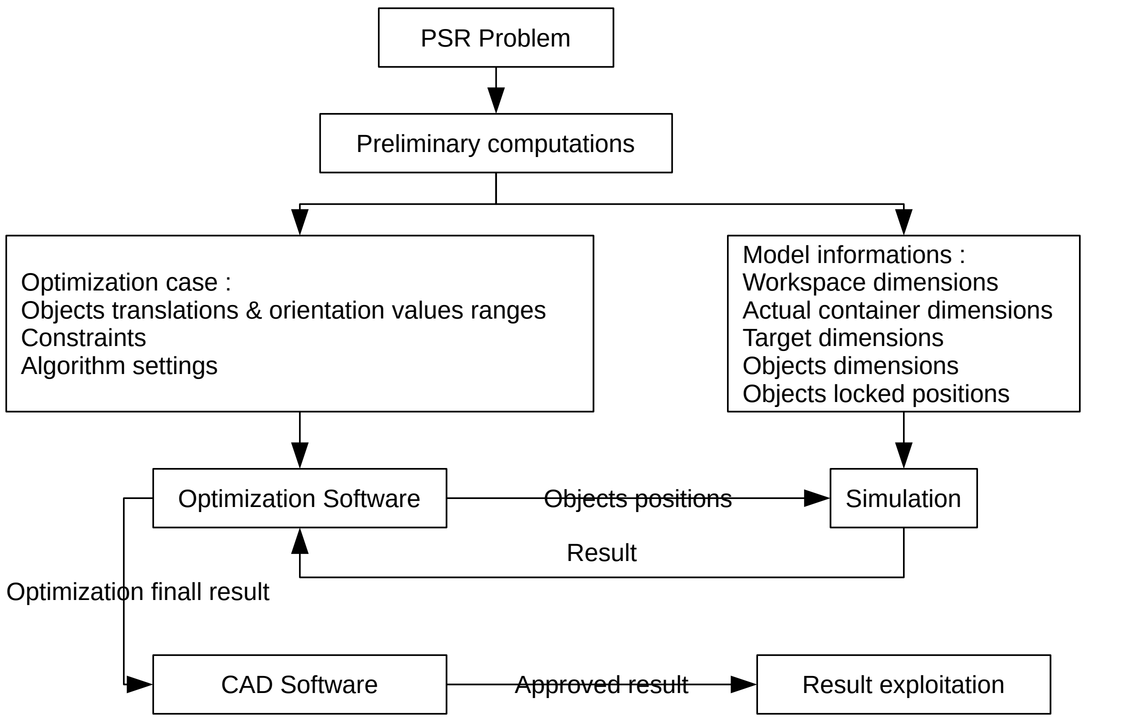


Figure 6: Approach overview

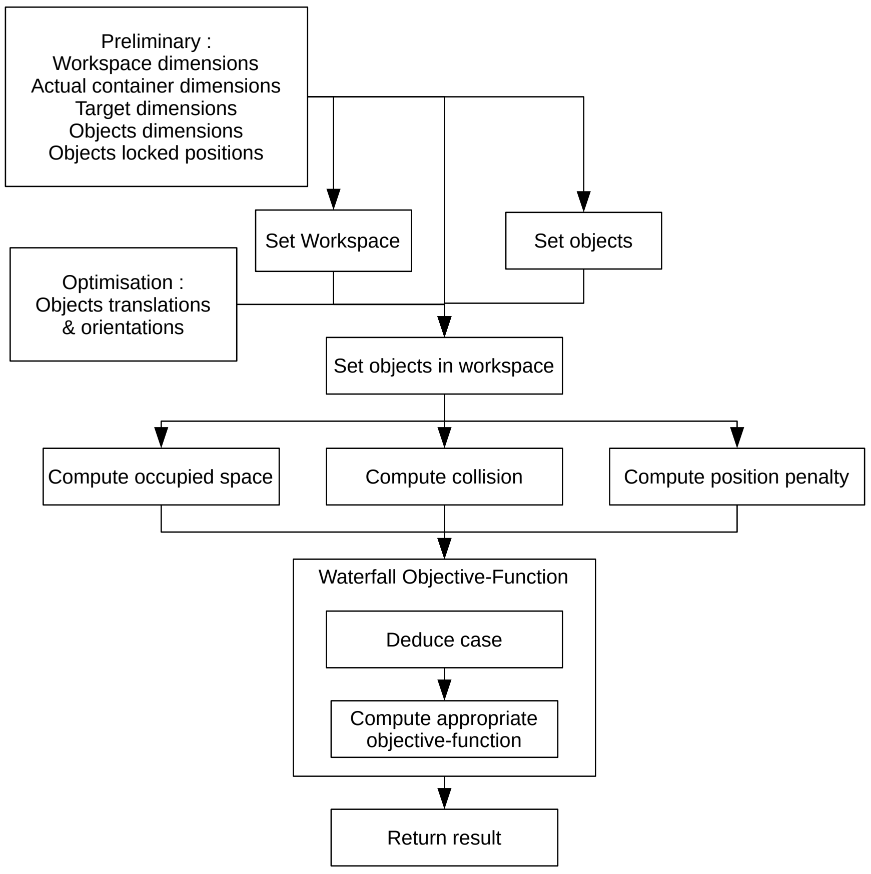


Figure 7: Simulation flowchart

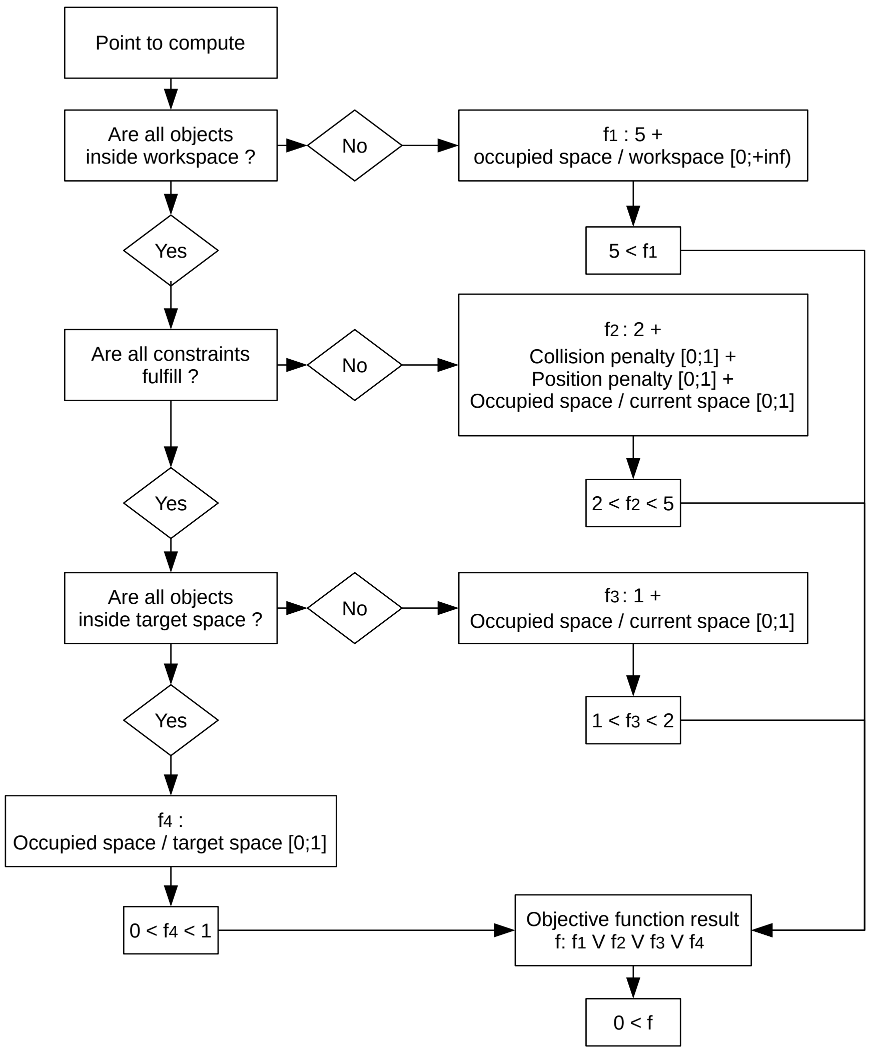

Point to compute
Are all objects inside workspace ?
No
f1 : 5 + occupied space / workspace [0;+inf)
Yes
5 < f1
Are all constraints fulfill ?
No
f2 : 2 + Collision penalty [0;1] + Position penalty [0;1] + Occupied space / current space [0;1]
Yes
2 < f2 < 5
Are all objects inside target space ?
No
f3 : 1 + Occupied space / current space [0;1]
Yes
1 < f3 < 2
f4 : Occupied space / target space [0;1]
0 < f4 < 1
Objective function result f: f1 V f2 V f3 V f4
0 < f

Figure 8: Waterfalls objective-function

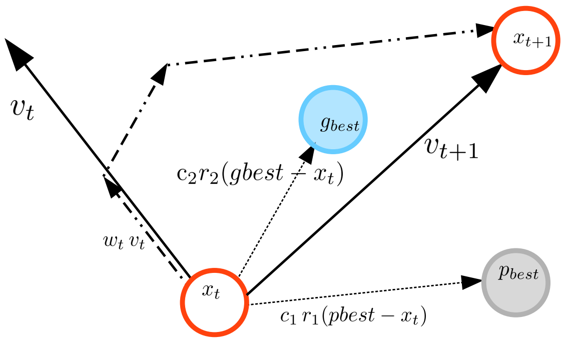


Figure 9: P.S.O. displacement

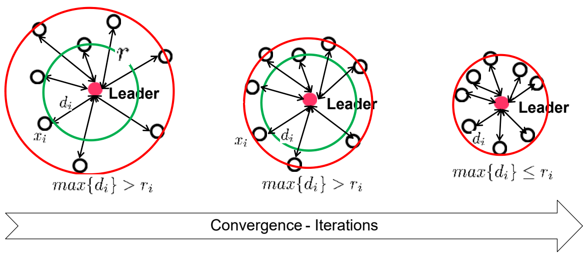


Figure 10: P.S.O. radius

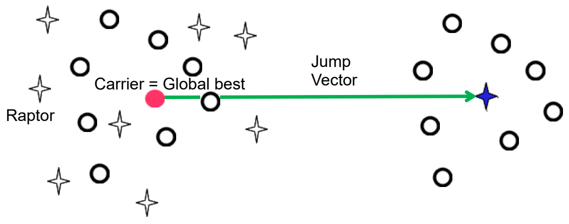
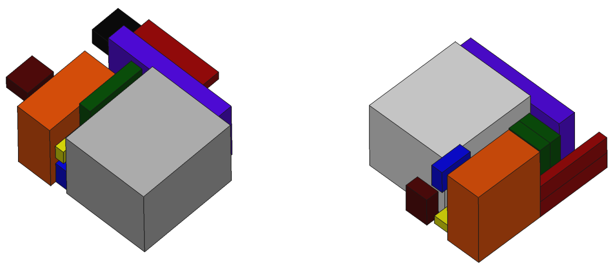


Figure 11: P.S.O. starcraft



(a) Before (b) After

Figure 12: Object positions