2013 IEEE Congress on Evolutionary Computation
June 20-23, Cancún, México

Multi-drop Container Loading using a Multi-objective Evolutionary Algorithm

Travis Kirke and Lyndon While
School of Computer Science & Software Engineering
The University of Western Australia
Perth, Australia 6009
Email: travis@kirke.com.au, lyndon@csse.uwa.edu.au

Graham Kendall
School of Computer Science
University of Nottingham
UK and Malaysia
Email: graham.kendall@nottingham.edu.my

***Abstract*—We describe a new algorithm MOCL (multi-objective container loading) for the multi-drop single container loading problem. MOCL extends the recent biased random-key genetic algorithm due to Goncalves & Resende to the multi-drop problem by enhancing its genetic representation, its fitness calculations, and its initialisation procedure. MOCL optimises packings both for volume utilisation and for the accessibility of the packed objects, by minimising the number of objects that block each other relative to a pre-defined unpacking schedule. MOCL derives solutions that are competitive with state-of-the-art algorithms for the single-drop case (where blocking is irrelevant), plus it derives solutions for 2–50 drops that give very good utilisation with no or very little blocking. This flexibility makes MOCL a useful tool for a variety of 3D packing applications.**

***Keywords:* evolutionary algorithms, multi-objective optimisation, cutting & packing.**

I. Introduction

The *single container loading problem* is a three-dimensional optimisation problem from the larger field of cutting & packing problems. It involves packing a set of objects into a large rectangular box such that the packing optimises some criterion or criteria. Real-world applications of this problem occur widely in many industries: for example instances in the transport industry arise in packing shipping containers, in packing items onto railway pallets, and in the removals industry. Instances of container loading problems vary with the shape and number of the objects to pack; with the heterogeneity of the set of objects; with constraints on the orientations of the objects; with whether the objects must be completely supported or not; and on other axes as well. [1] gives a comprehensive typology of the field.

By far the most common criterion used to assess packings is to maximise the volume utilisation of the container. However in many applications other criteria are also important, for example the accessibility of the packed objects relative to some pre-defined ordering (so-called *multi-drop problems*); the grouping of subsets of the objects that need to unpacked together; or the distribution of the objects inside the container, to ensure a good load distribution. Optimising for multiple criteria requires an algorithm to return a set of solutions offering different trade-offs between the various objectives, from which a client can select later depending on their priorities at the time[2].

The principal contribution of this paper is an implementation of a biased random-key genetic algorithm (BRKGA) *MOCL* (Multi-Objective Container Loading) that generates solutions for the multi-drop container loading problem optimised according to two objectives:

- it maximises the volume utilisation of the container;
- it maximises the accessibility of the packed objects, by minimising the number of objects that block each other relative to a pre-determined unpacking schedule.

MOCL is based on the recent BRKGA due to Goncalves & Resende[3], which gives the best current results for optimising volume utilisation alone. MOCL extends the genetic representation used in BRKGA to accommodate the more-general multi-drop problem, it extends the fitness calculations used in BRKGA to optimise for the extra criterion, and it extends the initialisation procedure used in BRKGA to get better diversity in the larger fitness space.

Experimental results on the widely-used problem set due to Bischoff & Ratcliff[4] show that MOCL generates solutions that are competitive with previous state-of-the-art approaches for the single-drop case (where blocking is irrelevant), plus it generates solutions for 2–50 drops that have no or very little blocking while still showing very good volume utilisation, and many other solutions in between. This flexibility makes MOCL a useful tool for a variety of 3D packing applications.

The rest of this paper is organised as follows. Section II defines the precise version of the container loading problem that we address. Section III describes previous approaches to similar problems, and also gives a short background on multi-objective optimisation. Section IV describes Goncalves & Resende's BRKGA for the container loading problem, plus our extensions to their algorithm for the multi-drop case. Section V reports experimental results obtained with MOCL, and compares its performance with previous approaches. Section VI concludes the paper and discusses some possibilities for future work.

II. Problem definition

The precise version of the container loading problem that we address in this paper extends the problem addressed in many other recent papers[5], [6], [7], [8], [9], [10], [11], [3]. A given 3D rectangular container is to be packed with a subset of a

given set of rectangular cuboids such that all packed boxes are completely inside the container, their sides are parallel to the walls of the container, and no boxes overlap. The rectangular container is specified simply by its physical dimensions, whilst the set of boxes is specified as a set of k box types, each comprising N_k identical objects specified by their physical dimensions. In addition, of the six spatial orientations available for each box, up to five of these can be prohibited: for example, some boxes may require that one side is always uppermost, whilst others may be unconstrained or only partly-constrained in this regard.

Another option is the *stability constraint*. If this is enforced, the bottom side of all packed boxes must be completely supported by either the floor of the container or the top side(s) of one or more other boxes. If it is not enforced, then any non-zero part of the bottom side of a packed box can be so supported. The stability constraint can be enforced or not as a parameter to MOCL.

We extend this problem to the multi-drop case by labeling each object with a drop number, consistently with Christensen & Rousoe[12]. Different drop numbers correspond to different customers: so all boxes addressed to Customer 1 must be unloaded first, then all boxes addressed to Customer 2 must be unloaded, and so on. Any time that a box addressed to Customer k impedes access to a box addressed to Customer $j < k$, that counts as a *block* and it degrades the value of our second objective. Fig. 1 illustrates how blocking is defined in MOCL, consistent with the definition used by Christensen & Rousoe.

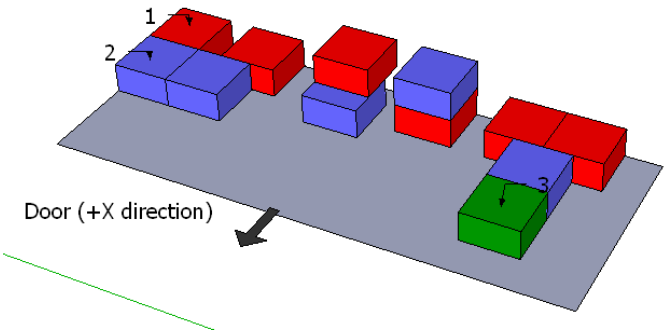


Fig. 1: Blocking in object placements. Each of the darker objects labeled 2 blocks each of the lighter objects labeled 1 that is either behind or below it, and the light object labeled 3 blocks both the 2 and the 1s behind it.

III. Background Material

This section briefly discusses previous recent approaches to the container loading problem, and it introduces some key concepts used in multi-objective optimisation.

A. Previous Container-loading Algorithms

The container loading problem is NP-hard[13], so only a few exact methods have been described[14]. Most approaches use heuristic procedures: we follow Fanslau & Bortfeldt[11] in classifying them according to the packing heuristic and the method type used. Table I lists significant recent approaches.

TABLE I: Previous recent approaches to the container loading problem (adapted from Table 1 in [3]).

Approach	Source	Classification
T_BB	Terno *et al.*[5]	Branch and bound
BG_GA	Bortfeldt & Gehring[6]	GA
BG_PGA	Bortfeldt & Gehring[7]	Parallel GA
E_TRS	Eley[8]	Tree search
L_GH	Lim *et al.*[15]	Greedy heuristic
B_PTS	Bortfeldt *et al.*[16]	Parallel tabu search (TS)
B_NMP	Bischoff[11]	Nelder-Mead procedure
M_SATS	Mack *et al.*[17]	Parallel SA/TS
MO_GR	Moura & Oliveira[10]	GRASP
P_GR	Parreno *et al.*[18]	GRASP
P_VNS	Parreno *et al.*[19]	Neighbourhood search
FB_TRS	Fanslau & Bortfeldt[11]	Tree search
HH_HBS	He & Huang[20]	Heuristic beam search
BRKGA	Goncalves & Resende[3]	Biased random-key GA
CR	Christensen & Rousoe[12]	Tree search

1) Packing Heuristics:

1) *Wall-building approaches* build vertical layers of objects, for example [6].
2) *Stack-building approaches* build stacks of objects arranged to save floor space, for example [4].
3) *Horizontal layer-building approaches* build horizontal layers of objects, designed to cover as much of each load surface as possible, for example [5].
4) *Block-building approaches* build cuboid blocks of objects, for example [8], [16], [17].
5) *Guillotine-cutting approaches* segment the container and fill the segments independently, for example [21].

2) Solution Methods:

1) *Meta-heuristics* are probably the most popular methods: they include tabu search, simulated annealing, genetic algorithms, and others, for example [6], [7], [16], [11], [17], [10], [18], [3].
2) *Tree-search or graph-search methods* have also been used, for example [21], [8], [11], [12].
3) *Conventional heuristics* incorporate construction methods and improvement methods, for example [4], [15].

B. Multi-objective Optimisation

In a multi-objective optimisation problem, potential solutions are assessed according to two or more independent quantities. The characteristic of good solutions is that improving in one objective can be achieved only by worsening in at least one other objective. An algorithm for solving such problems returns a set of solutions offering different trade-offs between the various objectives.

Consider a problem where the fitness function maps a solution x into a fitness vector $\overline{f_x}$. A solution x *dominates* a solution y iff $\overline{f_x}$ is at least as good as $\overline{f_y}$ in every objective, and is better in at least one objective. x is *non-dominated* wrt a set of solutions X iff there is no solution in X that dominates

x. X is a *non-dominated set* iff every solution in X is non-dominated wrt X. The set of fitness vectors corresponding to a non-dominated set is a *non-dominated front*.

x is *Pareto optimal* iff x is non-dominated wrt the set of all possible solutions. Such a solution is characterised by the fact that improvement in one objective can come only at the expense of some other objective(s). The *Pareto optimal set* is the set of all Pareto optimal solutions. The goal in multi-objective optimisation is to find (or approximate) this Pareto optimal set.

Having multiple objectives means there is only a partial order on solutions, which causes problems for selection in an evolutionary algorithm. The usual solution is to define a ranking on solutions: one popular scheme[22] defines the *rank* of a solution x wrt a set X to be the number of solutions in X that dominate x. Selection is then based on ranks: a lower rank implies a better solution.

Precise definitions of all these terms can be found in [2].

IV. The packing algorithm

This section describes the biased random-key genetic algorithm of Goncalves & Resende[3], and the ways in which we extend their algorithm for the multi-drop case.

A. Goncalves & Resende

For a problem with M boxes, BRKGA uses a chromosome with $2M$ genes.

- The first M genes specify the *box type packing sequence* using a random-key approach[23]: there is one gene in $[0, 1]$ for each individual box, and the genes are sorted to derive the packing order, as shown in Fig. 2. Mutation of these values allows the chromosome to represent different packing orders. The principal advantage of this approach is that all possible chromosomes are feasible solutions.

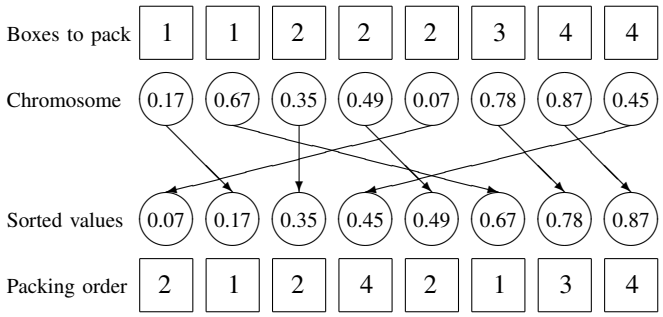


Fig. 2: The chromosome decoding procedure for the box type packing sequence in BRKGA (adapted from Fig. 3 in [3]).

- The other M genes specify the *vector of layer types*. Each box can have up to 6 orientations, and BRKGA allows 6 different layer types, as shown in Fig. 3. Together these form 6 $\times$ 6 = 36 possibilities, and each gene is an integer in $[1, 36]$ which is used as an index into this list.

BRKGA turns a chromosome into a packing by an iterative procedure, with five steps in each iteration. Throughout the process it maintains state information about the boxes remaining to be packed, and the empty spaces available in the container.

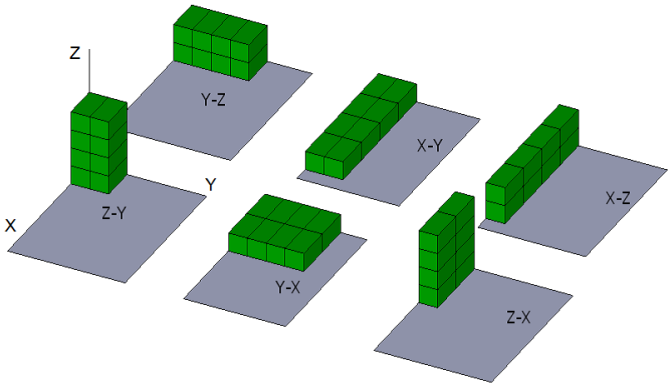


Fig. 3: The six different layer types in BRKGA.

1) From the box type packing sequence it identifies the next box type k^* to pack.
2) It uses a back-bottom-left procedure to identify the best space EMS^* available for packing boxes of type k^*.
3) From the vector of layer types it selects a layer type L^* suitable for packing boxes of type k^* into EMS^*.
4) It packs as many boxes of type k^* as it can fit into EMS^* using layer type L^*.
5) It updates its state information, removing the boxes that were packed in this iteration and updating the list of empty spaces in the container.

The process finishes when no more boxes can be packed into the container. The volume utilisation is simply the total volume of the packed boxes divided by the volume of the container.

Further details on BRKGA, for example on the evolutionary process used, can be found in [3].

B. Extension to MOCL

Apart from introducing the framework necessary for performing multi-objective optimisation (principally the calculation of ranks as described in Section III-B and their use in selection), MOCL extends the BRKGA algorithm in three principal ways: we enhance the representation, the fitness calculations, and the initialisation procedure.

1) Representation: In each iteration, BRKGA packs as many boxes of the current type as fit into the space chosen. This works well when maximising volume utilisation is the only goal, but when blocking is added as a second objective, we need finer control over this part of the procedure: often we want to pack only boxes addressed to one (or a small number) of the customers.

Thus we add to the chromosome another M genes, each in $[0, 1]$ which specify for each box the maximum proportion of that box type that can be packed in one iteration. High values of these genes will usually give good volume utilisation: other values will often give good blocking performance.

TABLE II: Volume utilisation of various recent packing algorithms for the single-drop case when the support constraint is enforced. Each value is a percentage of the container's volume. The first eight columns of data are taken from Table 4 in [3]; the ninth column is taken from Table 11.2 in [12].

	T_BB[5]	BG_GA[6]	BG_PGA[7]	E_TRS[8]	B_NMP[9]	MO_GR[10]	FB_TRS[11]	BRKGA[3]	CR[12]	MOCL
BR1	89.9	87.81	88.10	88.0	89.39	89.07	94.51	94.34	90.48	93.53
BR2	89.6	89.40	89.56	88.5	90.26	90.43	94.73	94.88	91.16	93.71
BR3	89.2	90.48	90.77	89.5	91.08	90.86	94.74	95.05	91.33	93.76
BR4	88.9	90.63	91.03	89.3	90.90	90.42	94.41	94.75	90.98	93.58
BR5	88.3	90.73	91.23	89.0	91.05	89.57	94.13	94.58	90.83	93.33
BR6	87.4	90.72	91.28	89.2	90.70	89.71	93.85	94.39	90.19	92.94
BR7	86.3	90.65	91.04	88.0	90.44	88.05	93.20	93.74	88.36	92.12
average	88.51	90.06	90.43	88.79	90.55	89.73	94.22	94.53	90.48	93.28

2) Objectives: We augment the fitness calculations of BRKGA with the second objective of minimising blocking relative to a pre-defined unpacking schedule. Each box is randomly assigned a drop number as described in Section II, and we say that Object A obstructs Object B if A has a higher drop number than B and

- A is on top of B (i.e. the base of A is higher than the top of B and they overlap horizontally), or
- A is in front of B (i.e. the back of A is in front of the front of B, and they overlap across the container), or
- A obstructs Object C, and C obstructs B.

The blocking value for a packing is simply the count of all such obstructions.

3) Initialisation: Clearly packings are always possible that give zero blocking: simply ensure that the packed boxes are ordered by decreasing drop number. However it can be hard to evolve these packings naturally, especially for large numbers of drops. To ensure that MOCL gives good performance in this regard, we pre-construct chromosomes for such solutions and use them to seed part of the initial population.

V. Experimental results

We assess the performance of MOCL in three ways.

- We compare its volume utilisation for the single-drop problem with Goncalves & Resende's biased random-key GA[3] and with a number of other recent algorithms.
- We compare its volume utilisation for the multi-drop problem with 2–50 drops with Christensen & Rousoe's algorithm[12], both for 0-blocking solutions and for "low-blocking solutions" with 0.5–2 blocks/drop.
- We look at the Pareto fronts returned by MOCL, to further illustrate the trade-offs available via the multi-objective approach.

The comparison reported here is over the BR1–7 problem sets from the OR library[4] with the stability constraint enforced, because our principal focus is on the comparison with Christensen & Rousoe. (Other results are reported in [24].) The seven data sets have increasing levels of heterogeneity, with 3, 5, 8, 10, 12, 15, and 20 box types respectively. Each set comprises 100 separate problems with randomised numbers of boxes, dimensions, and orientation constraints. Each individual problem has 100–250 boxes in total, the sum of whose volume does not exceed the volume of the given container: thus it is always possible (although not guaranteed) for there to be a perfect solution.

To extend these data sets for the multi-drop problem, we take a number of customers 2, 5, 10, or 50, and assign each box uniformly randomly to one of these customers. We note two features of the assignment of boxes to customers.

- The assignment is done independently for each number of drops. Thus there is no correspondence between the assignments for a particular problem with, say, five drops and ten drops. In particular, a good solution for the latter cannot be turned into a good solution for the former simply by combining pairs of customers, or by any other transformation, and *vice versa*.
- The assignments used here are different to the assignments used by Christensen & Rousoe, as their data is unavailable. However, given that each data set has 100 problems and that we report only averages, this is unlikely to introduce any systematic bias in the results.

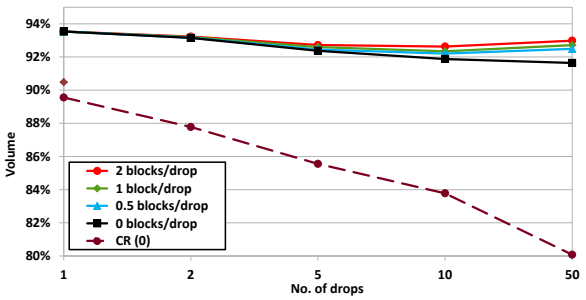
MOCL was implemented in Java v1.6, and the experiments were run on systems with Intel Core i3 dual-core CPUs and 4Gb of RAM, running Red Hat Linux. Typical running times for MOCL are about five minutes for the supported 1-drop case, and about ten minutes for the supported multi-drop cases. These times are somewhat higher than some other recent algorithms, which is at least partly due to differences in the language used and in the level of parallelisation employed.

A. Volume utilisation

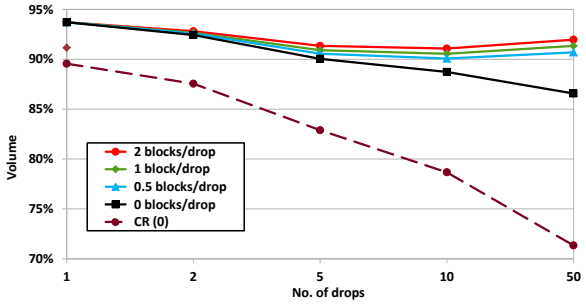
Table II shows the volume utilisation for MOCL and for a selection of recent algorithms from the literature. The figure shown for CR is for the version of their algorithm that is optimised for the single-drop problem, where blocking does not arise.

MOCL shows competitive results with the state-of-the-art algorithms FB_TRS and BRKGA (on which it is based), although it consistently returns figures 1–1.5% lower than BRKGA. This difference is probably due to the multi-objective nature of the new problem, which will tend to disperse the population over the larger fitness space.

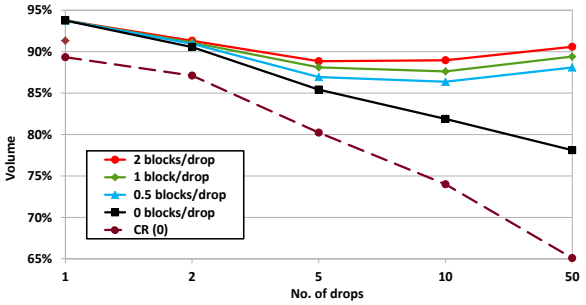
For every data set, MOCL returns figures around 2.5–3% higher than CR, the only other algorithm that optimises for blocking.



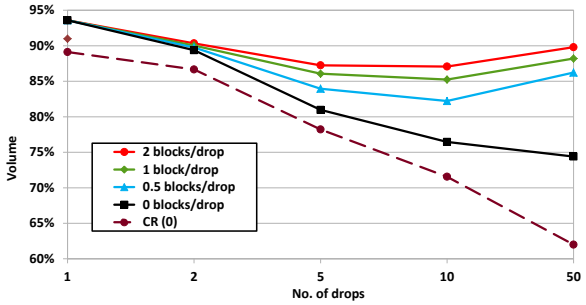
(a) BR1



(b) BR2



(c) BR3



(d) BR4

Fig. 4: Volume utilisation of MOCL and CR in low-blocking situations for BR1–4. Each graph shows the 0-blocking solutions and the optimised 1-drop solution for CR, the 0-blocking solutions for MOCL, and solutions for MOCL with 0.5–2 blocks/drop. Note the varying scales on the y-axis.

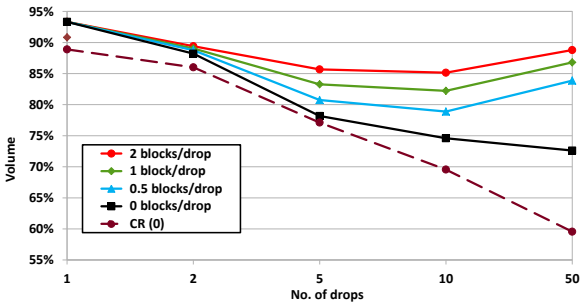
B. Low-blocking solutions

Figs. 4 and 5 show the 0- and low-blocking solutions returned by MOCL and CR. A low-blocking solution with k blocks/drop for d drops is allowed to have up to kd blocks over the entire packing, although the 100 problems in each data set are treated separately. We note however that usually the level of blocking is well below the limit:

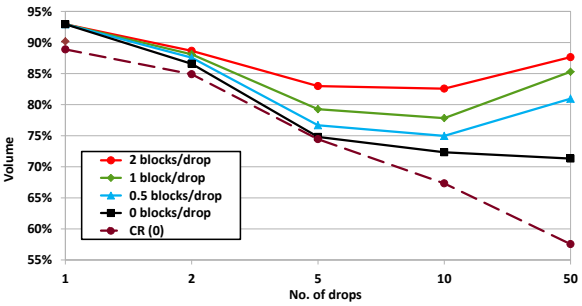
- with 0.5 blocks/drop allowed, the number of blocks/drop actually present averages only 0.24, with a maximum value of 0.31;
- with 1 block/drop allowed, the average present is only 0.59, with a maximum of 0.74;
- with 2 blocks/drop allowed, the average present is only 1.34, with a maximum of 1.65.

Note that with say 0.5 blocks/drop, on average half of the drops will involve no blocking at all.

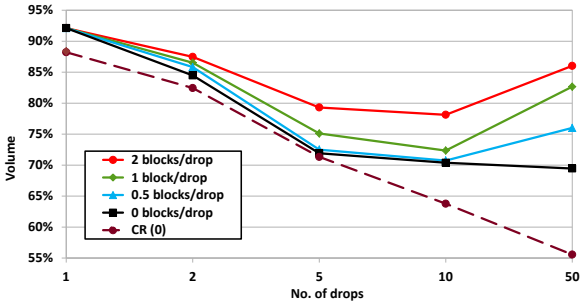
For 0-blocking, MOCL returns figures that are always higher than CR, typically by around 3–12%, although CR gets closer for 2 and 5 drops on the harder (i.e. more heterogeneous) problem sets. The multi-objective nature of MOCL however means that it also naturally generates solutions for other blocking numbers, and the graphs show that allowing an average of only 0.5 blocks/drop allows MOCL to achieve volume utilisations often over 10% higher than its own 0-blocking solutions. These improvements are greatest for the harder problem sets (for the easier problem sets MOCL is



(a) BR5



(b) BR6



(c) BR7

Fig. 5: Volume utilisation of MOCL and CR in low-blocking situations for BR5–7. Each graph shows the 0-blocking solutions and the optimised 1-drop solution for CR, the 0-blocking solutions for MOCL, and solutions for MOCL with 0.5–2 blocks/drop.

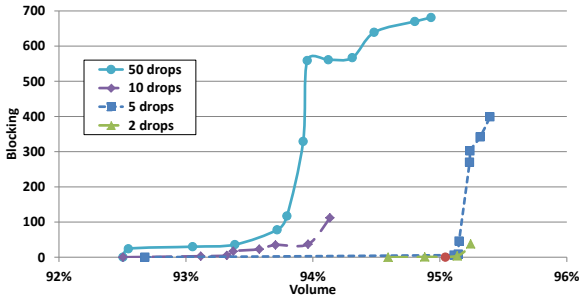
already operating close to capacity) and for higher numbers of drops. Note that as the number of drops increases, MOCL has increased opportunity to amortise the permitted blocks over the drops: this flexibility contributes to the very high level of performance on fifty drops. Other experiments have shown that there are diminishing returns as the permitted number of blocks/drop increases beyond about 4.

C. *Trade-off fronts*

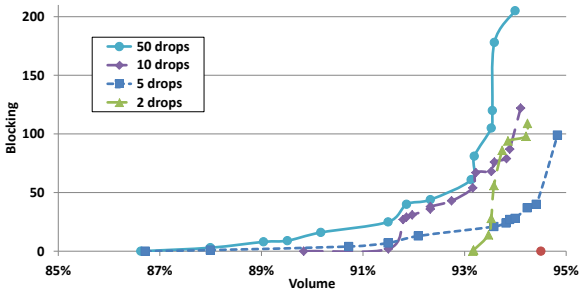
Fig. 6 shows the Pareto fronts returned by MOCL for selected individual problems. The graphs show that MOCL returns multiple solutions for most problems and most numbers of multiple drops, offering different trade-offs between the two objectives from which a client can choose later depending on their priorities at the time. The average front-size for each problem varies from 8 solutions up to about 38.

Unless the client heavily prioritises one of the objectives over the other, most interesting will often be where a front has a "knee" which indicates a non-linear trade-off between the objectives.

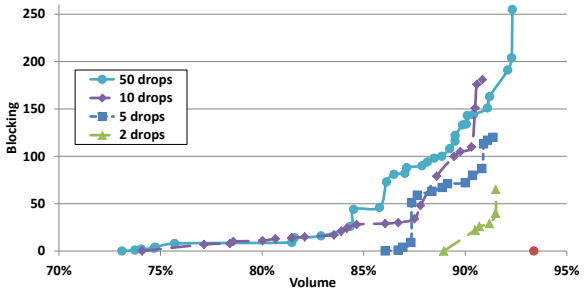
- Many of the lines on the graphs have convex knees, for example the 5-drop line for BR2/50, the 50-drop line for BR3/1, and the 10-drop line for BR5/63. In such a case the client will normally prefer the solutions at the knee: moving away from the knee in either direction will offer a poor trade-off, sacrificing a lot in one objective for only a small improvement in the other. For example, on the 5-drop line for BR2/50, the choice is between a knee with 95.1% utilisation with 6 blocks, against 92.7% utilisation



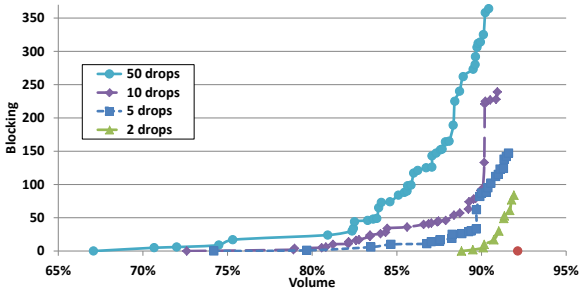
(a) BR2/50



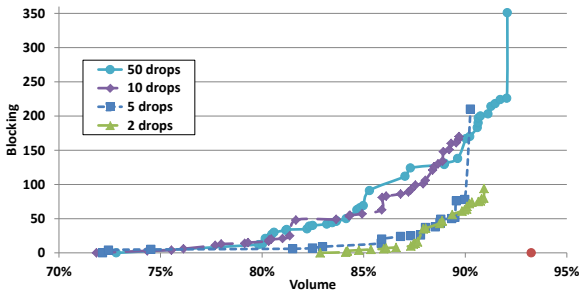
(b) BR3/1



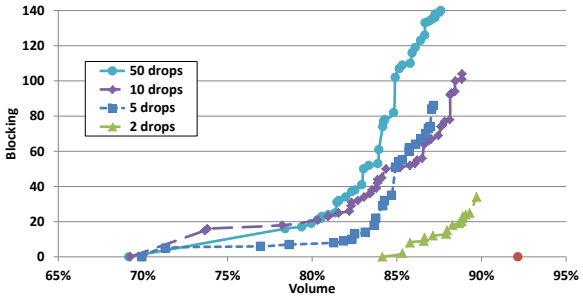
(c) BR4/11



(d) BR5/63



(e) BR6/80



(f) BR7/99

Fig. 6: Pareto fronts for selected problems. Each graph shows all solutions returned by MOCL for the named problem, in objective space. Note the varying scales on both axes.

with 0 blocks in one direction and 95.2% utilisation with 46 blocks in the other direction. Clearly the first is the best trade-off.

- Some of the lines on the graphs show concave knees, for example the 50-drop line for BR2/50 (the mid-part of the line from 100–600 blocks), the 5-drop line for BR4/11 (centred around 50 blocks), and the 2-drop line for BR3/1 (centred around 90 blocks). In the concave case the client will normally prefer the solutions away from the knee: moving away in either direction will offer a good trade-off. For example, on the 50-drop line for BR2/50, the choice is between a knee with 94.0% utilisation with 559 blocks, against 93.8% utilisation with 117 blocks in one direction and 94.3% utilisation with 567 blocks in the other direction. Clearly the first is the worst trade-off.
- Many of the lines show the greatly improved utilisation that can be achieved by allowing small amounts of blocking. For example, the 5-drop line for BR7/99 shows 69.9% utilisation with 0 blocks, against 82.4% utilisation with 10 blocks (i.e. 2 blocks/drop); and the 10-drop line for BR6/80 shows 71.8% utilisation with 0 blocks, against 80.3% utilisation with 17 blocks (i.e. 1.7 blocks/drop).

It is crucial to note that these patterns are exposed as a *result* of the algorithm: the user of MOCL does not need to specify anything in advance. This is a major strength of the multi-objective approach.

As stated earlier, there is no correspondence between the customer assignments for different numbers of drops for a given problem, so sometimes a problem with $k > 1$ drops will have an "easier" assignment than the same problem with $1 < j < k$ drops. However clearly every assignment to k drops is related to the single-drop version of the same problem: thus in the (unusual) event that the k-drop version returns a volume utilisation better than the single-drop version, we could use that packing and its performance to improve MOCL's results. However we have not done this in the results reported here, because it would significantly affect the time performance of MOCL, and we cannot predict when it will occur.

VI. Conclusions

We have described a new algorithm MOCL for the multi-drop single container loading problem. MOCL optimises solutions according to two objectives: volume utilisation, and the accessibility of the objects relative to a pre-defined unpacking schedule. MOCL extends Goncalves & Resende's state-of-the-art biased random-key genetic algorithm to the multi-drop case, by extending the genetic representation, the fitness calculations, and the initialisation procedure. MOCL returns competitive results for the single-drop case (where blocking is irrelevant), plus it generates solutions for 2–50 drops that have no or very little blocking while still showing very good volume utilisation, and many other solutions in between. This flexibility makes MOCL a useful tool for a variety of 3D packing applications.

In the future we plan to improve MOCL by refining the initialisation procedure further, possibly by relating together instances of a given problem for different drop numbers. There are probably also other features of BRKGA that could be re-designed to work better for the multi-drop case.

References

[1] G. Wäscher, H. Hauβner, and H. Schumann, "An improved typology of cutting and packing problems," *European Journal of Operational Research*, vol. 183, pp. 1109–1130, 2007.

[2] C. Coello Coello, G. Lamont, and D. Van Veldhuizen, *Evolutionary Algorithms for Solving Multi-objective Problems*. Springer, 2007.

[3] J. F. Goncalves and M. G. C. Resende, "A parallel multi-population biased random-key genetic algorithm for a container loading problem," *Computers & Operations Research*, vol. 39, pp. 179–190, 2012.

[4] E. E. Bischoff and M. S. W. Ratcliff, "Issues in the development of approaches to container loading," *Omega*, vol. 23, pp. 377–390, 1995.

[5] J. Terno, G. Scheithauer, U. Sommerweiss, and J. Riehme, "An efficient approach for the multi-pallet loading problem," *European Journal of Operational Research*, vol. 123, pp. 372–381, 2000.

[6] A. Bortfeldt and H. Gehring, "A hybrid genetic algorithm for the container loading problem," *European Journal of Operational Research*, vol. 131, pp. 143–161, 2001.

[7] ——, "A parallel genetic algorithm for solving the container loading problem," *International Transactions in Operational Research*, vol. 9, pp. 497–511, 2002.

[8] M. Eley, "Solving container loading problems by block arrangement," *European Journal of Operational Research*, vol. 141, pp. 393–409, 2002.

[9] E. Bischoff, "Three-dimensional packing of items with limited load bearing strength," *European Journal of Operational Research*, vol. 168, pp. 952–966, 2006.

[10] A. Moura and J. Oliveira, "A GRASP approach to the container loading problem," *IEEE Intelligent Systems*, vol. 20, pp. 50–57, 2005.

[11] T. Fanslau and A. Bortfeldt, "A tree search algorithm for solving the container loading problem," *INFORMS Journal of Computing*, vol. 22, pp. 222–235, 2010.

[12] S. G. Christensen and D. M. Rousoe, "Container loading with multi-drop constraints," Masters dissertation, Institute of Informatics and Mathematical Modelling, Technical University of Denmark, 2007.

[13] G. Scheithauer, "Algorithms for the container loading problem," *Operations Research Proceedings 1991*, pp. 445–452, 1992.

[14] S. Martello, D. Pisinger, and D. Vigo, "The three-dimensional bin packing problem," *Operations Research*, vol. 48, pp. 256–267, 2000.

[15] A. Lim, B. Rodrigues, and Y. Wang, "A multi-faced buildup algorithm for three-dimensional packing problems," *Omega*, vol. 31, pp. 471–481, 2003.

[16] A. Bortfeldt, H. Gehring, and D. Mack, "A parallel tabu search algorithm for solving the container loading problem," *Parallel Computing*, vol. 29, pp. 641–662, 2003.

[17] D. Mack, A. Bortfeldt, and H. Gehring, "A parallel hybrid local search algorithm for the container loading problem," *International Transactions in Operational Research*, vol. 11, pp. 511–533, 2004.

[18] F. Parreno, R. Alvarez-Valdes, J. Tamarit, and J. Oliveira, "A maximal-space algorithm for the container loading problem," *INFORMS Journal of Computing*, vol. 20, pp. 412–422, 2008.

[19] F. Parreno, R. Alvarez-Valdes, J. Oliveira, and J. Tamarit, "Neighbourhood structures for the container loading problem: a vns implementation," *Journal of Heuristics*, vol. 16, pp. 1–22, 2010.

[20] K. He and W. Huang, "Solving the single-container loading problem by a fast heuristic method," *Optimization Methods and Software*, vol. 25, pp. 263–277, 2010.

[21] R. Morabito and M. Arenales, "An and/or-graph approach to the container loading problem," *International Transactions in Operational Research*, vol. 1, pp. 59–73, 1994.

[22] C. M. Fonseca and P. J. Fleming, "Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization," in *Proceedings of the Fifth International Conference on Genetic Algorithms*, S. Forrest, Ed. Morgan Kaufmann Publishers, 1993, pp. 416–423.

[23] J. C. Bean, "Genetics and random keys for sequencing and optimization," *ORSA Journal on Computing*, vol. 6, pp. 154–160, 1994.

[24] T. Kirke, "A genetic algorithm approach to the multi-drop container loading problem," Honours dissertation, School of Computer Science & Software Engineering, The University of Western Australia, 2012.