Solving the Casting problem using Column Generation; better results with 100 variables instead of 1 billion

Jippe Hoogeveen[1][0000−0002−9554−9316], Marjan van den Akker[1][0000−0002−7114−0655], and Han Hoogeveen[1][0000−0001−8544−8848]

Department of Information and Computing Sciences, Utrecht University, Princetonplein 5, 3584 CC Utrecht, The Netherlands jippehoogeveen1@gmail.com; J.M.vandenAkker@uu.nl; J.A.Hoogeveen@uu.nl

Abstract. In casting problems the goal is to cast a number of orders such that the number of heats needed to obtain the molten metal for the casting is minimized. In a paper by Deb and Myburgh from 2016 the authors presented a Genetic Algorithm (GA) to find a near-optimal solution, which took their GA less than 150 hours for the largest instances with up to 100 million heats. They further stated that this type of problems could not be solved using Integer Linear Programming (ILP) for problem instances with 200 heats or more. In this paper, we refute their claim by showing that an ILP formulation based on a set covering formulation that we solve using Column Generation can easily find almost optimal solutions (at most a few heats more than the theoretical minimum) within a second, even for their instances requiring 100 million heats. We further show that this ILP model can easily be extended to deal with multiple crucibles and restricted availability of the orders due to release dates and deadlines.

Keywords: Casting problem · crucibles · cutting stock · Linear Programming · Column Generation · deadlines.

1 Introduction

In this paper we consider the basic casting problem that was presented by Deb and Myburgh (2016) and the more elaborated versions discussed in Deb et al. (2003). In our description of the problem below, we consider the most elaborate version, but we handle the several variants in order of difficulty, starting with the basic problem in Section 2.

We are given a foundry that operates one or two crucibles (depending on the variant) to melt metal. The two crucibles have a different capacity, which we denote by W_1 and W_2. Per day, we can use the crucibles m_1 and m_2 times, respectively. The molten metal is used to cast objects that have been ordered by customers. We assume that we are given n orders; each order j ($j = 1, \ldots, n$) consists of a request for r_j ($j = 1, \ldots, n$) identical copies, where each copy requires an amount of w_j units of molten material. For each order j ($j = 1, \ldots, n$) we are given a deadline d_j at which the entire order must be ready; we further have added an *earliest start time* or *release date* e_j, before which we cannot start producing order j ($j = 1, \ldots, n$). The goal depends on the variant. If there is only one crucible available, then the obvious goal is to minimize the number of heats needed. In case of two (or more) crucibles, we assume that the goal is to finish as soon as possible, but in Sections 3 and 4 we will discuss some alternative objective functions as well.

Deb and Myburgh (2016) consider the basic problem with either one or two crucibles. An essential aspect of their approach is that they compute a target value H on the number of heats based on a desired efficiency metal utilization rate η, which has a typical value of 0.997 in their computational experiments. Next, they look for a solution that uses exactly H heats, thereby turning the optimization problem into a feasibility problem. To solve this feasibility problem they design a Genetic Algorithm and they further formulate it as an Integer Linear Program, which is solved using the commercial solver CPLEX. On basis of their computational experiments, they conclude that their Genetic Algorithm clearly outperforms their ILP approach.

In this paper we show that the disability of their ILP to find good solutions is due to the type of formulation that Deb and Myburgh (2016) use, and that in fact ILP techniques work far better than a Genetic Algorithm for this type of problem. Thereto, we present a different type of ILP formulation that is capable of finding solutions that are at most a few heats away from the theoretical optimum, which does not depend on the utilization rate η. We further show that this ILP can easily be extended to deal with the extensions mentioned in the paper by Deb et al. (2003), for which we again find solutions very close to the theoretical optimum in seconds.

2 ILP formulation

As mentioned above, Deb and Myburgh (2016) formulate the feasibility problem *'Does there exist a solution that uses H heats'* as an ILP (which we call the *DM-ILP*). Hereto, they use variables x_{jh} $(j = 1, \ldots, n; h = 1, \ldots, H)$ that indicate the number of copies of order j that are cast from heat h. This ILP is very hard to solve for an ILP solver because of the integrality of the x_{jh} variables, as Deb and Myburgh (2016) remark. Moreover, this ILP formulation suffers tremendously from *symmetry*, as all $H!$ solutions that can be obtained by renumbering the H heats are considered different by the ILP solver and hence might be investigated. Their ILP formulation has a constant objective (say 0) that must be minimized subject to the constraints

$$\sum_{h=1}^{H} x_{jh} \geq r_j \qquad \forall j = 1, \ldots, n \tag{1}$$

$$x_{jh} \geq 0 \text{ and integral} \qquad \forall j = 1, \ldots, n; \quad \forall h = 1, \ldots, H \tag{2}$$

Looking at the optimization variant of the problem as a *cutting stock* problem opens the way for a different ILP formulation. In the one-dimensional cutting stock problem we are given a set of identical bars with length L from which we must cut pieces to fulfill n orders. Here order j $(j = 1, \ldots, n)$ requests to cut r_j copies of length w_j; the goal is to minimize the amount of wasted material, which boils down to minimizing the number of bars that are cut. The cutting stock problem was already described by Kantorovich in 1939 (see Kantorovich, 1960), and it formed the inspiration for Gilmore and Gomory to establish the technique of *Column Generation*, (see Gilmore and Gomory, 1961, 1963, 1965).

The basic casting problem with only one crucible with capacity W and n orders of r_j copies of size w_j can easily be translated into a cutting stock problem: we have bars of length W, from which we are asked to cut r_j pieces of length w_j, for $j = 1, \ldots, n$. The goal is to minimize the number of heats used, which comes down to minimizing the number of bars used. Hence, we can formulate the basic casting problem as an ILP similarly to the ILP formulation of the cutting stock problem (which we call the *CG-ILP*). Thereto, we define *heat patterns*, which indicate how we use the W units of molten material from a single heat. Heat pattern s is described by the vector $(a_{s1}, a_{s2}, \ldots, a_{sn})$, where a_{sj} signals the number of copies that are cast from the object involved in order j $(j = 1, \ldots, n)$. Since we want our heat patterns to be feasible, each heat pattern s must satisfy the constraints

$$\sum_{j=1}^{n} w_j a_{sj} \leq W, \text{ and} \tag{3}$$

$$a_{sj} \geq 0 \text{ and integral} \quad \forall j = 1, \ldots, n. \tag{4}$$

Let S denote the set of feasible heat patterns. For each heat pattern $s \in S$ we introduce a variable x_s that indicates the number of heats that are used according to heat pattern $s \in S$. This leads to the following ILP formulation

$$\min \sum_{s \in S} x_s \qquad \text{subject to} \tag{5}$$

$$\sum_{s \in S} a_{sj} x_s \geq r_j \qquad \forall j = 1, \ldots, n \tag{6}$$

$$x_s \geq 0 \text{ and integral} \qquad \forall s \in S \tag{7}$$

Obviously, not all feasible heat patterns are known, and enumerating these is computationally intractable in general. Gilmore and Gomory (1961) have shown that a near-optimal solution can be obtained by the technique of *Column Generation*. Hereto, we first relax the integrality constraints to $x_s \geq 0$ for all $s \in S$. We start with only a small subset S' of S, for example the n basic heat patterns in which we produce as many copies as possible of object j ($j = 1, \ldots, n$) only. Next, we solve the resulting LP for the variables in S'; as a by-product we find the *shadow prices* π_j ($j = 1, \ldots, n$), where shadow price π_j corresponds to the constraint for order j in Constraint 6. We then generate heat patterns that improve the solution of the current LP; it is well known from LP-theory that this is only possible for heat patterns with negative *reduced cost*. The reduced cost of the heat pattern characterized by $(a_{s1}, a_{s2}, \ldots, a_{sn})$ is equal to

$$1 - \sum_{j=1}^{n} \pi_j a_{sj}.$$

To find a heat pattern with negative reduced cost (if it exists) we solve the so-called *pricing problem*, in which we search for the heat pattern s for which the reduced cost is minimized. Hence, we aim to find the heat pattern s for which $(a_{s1}, \ldots, a_{sn})$ maximizes

$$\sum_{j=1}^{n} \pi_j a_{sj}$$

subject to Constraints 3 and 4. This pricing problem corresponds to a variant of the KNAPSACK problem, where the knapsack has a volume of W and there are n items. Item j corresponds to product j ($j = 1, \ldots, n$); it has a revenue of π_j and a weight of w_j, and the variable a_{sj} defines the number of times that item j gets selected. This variant of the KNAPSACK problem can be solved in $O(nW)$ time by Dynamic Programming (see for example Kellerer et al., 2004). To do this, we use state variables $f_j(t)$ to denote the maximum profit of any knapsack of weight at most t that uses only items $1, 2, \cdots, j$. We use $j = 1, 2, \ldots, n$ and $t = 0, 1, \ldots, W$ and get the recursive relation

$$f_j(t) = \max\{f_{j-1}(t), f_j(t - w_j) + \pi_j\}$$

where the first alternative describes a knapsack without item j and the second describes a knapsack with at least one item j (note that this is only possible if $t \geq w_j$). By calculating $f_j(t)$ in order of j and t, we can calculate all $f_j(t)$ correctly in $O(nW)$ running time. Next, we find the optimal profit for this instance of KNAPSACK at $f_n(W)$, and we can find a solution with this value by backtracking through the DP table. When $f_n(W) > 1$, then this solution corresponds to a feasible heat pattern with negative reduced cost; we add it to S', solve the LP again, etc. When the minimum reduced cost is non-negative (which corresponds to $f_n(W) \leq 1$), then we have solved the LP-relaxation and the outcome value gives us a valid lower bound to the ILP.

Let x^* denote the solution to the LP-relaxation. If x^* is integral, then we have found an optimal solution; if there are variables in x^* with a fractional value, then we construct a feasible integral solution by rounding. The easiest way is by rounding up each fractional value. Since the number of variables in x^* with a positive value is at most equal to the number of constraints, which amounts to n, the difference between the outcome value of x^* and the outcome value of the rounded up solution is less than n, and hence, because of integrality, we use at most $n - 1$ heats more than the optimum solution. If we want to reduce this potential gap, then we can also round down x^*.

We then produce too few copies of some products, and the remaining copies must be cast using additional heats. This results in a much smaller instance of the basic casting problem. Since the feasible solution obtained by rounding x^* up uses at most n heats more than the solution in which we round x^* down, the lacking copies of the objects can be cast using at most n heats. Unless n is very big, this problem can be solved by using the DM-ILP in combination with binary search on H. An alternative might be to use some heuristic or metaheuristic.

We can also slightly change the DM-ILP to immediately find the optimal value of H. Recall that the original ILP formulation of Deb and Myburgh (2016) looked like:

$$\min 0 \qquad \text{subject to} \tag{8}$$

$$\sum_{j=1}^{N} w_j x_{i,j} \leq W \qquad \forall i = 1, \ldots, H \tag{9}$$

$$\sum_{i=1}^{H} x_{i,j} = r_j \qquad \forall j = 1, \ldots, n \tag{10}$$

$$x_{ij} \geq 0 \text{ and integral} \qquad \forall i = 1, \ldots, H \text{ and } j = 1, \ldots, n \tag{11}$$

We can slightly modify this ILP by adding binary variables y_i for all $i = 1, \ldots, H$ that denote if we want to use heat i (then $y_i = 1$). This changes the objective to

$$\min \sum_{i=1}^{H} y_i$$

and changes Constraint 9 to

$$\sum_{j=1}^{N} w_j x_{i,j} \leq W y_i \qquad \forall i = 1, \ldots, H$$

This ILP is able to find the minimal number of heats required in one go. It will generally be harder to solve than the ILP from Deb and Myburgh (2016) due to the extra binary variables, but as we only have a few remaining copies left, it should be doable. The advantage is that this ILP can be used more generally when we also have deadlines and release dates, where the binary search on H becomes impossible. We can reduce the amount of symmetry by adding the constraints

$$y_i \geq y_{i+1} \qquad \forall i = 1, \ldots, H-1. \tag{12}$$

If we know some lower bound L to the required number of heats (for example on basis of the required amount of molten iron), then we can add that $y_j = 1$ for all $j = 1, \ldots, L$. Finally, if we want to be sure that there is a feasible solution, then we can replace the maximum number of heats H with some upper bound on the number of heats that certainly allows a feasible solution. We call this ILP the *DM+-ILP*.

3 Working with two different crucibles

In the basic problem we have only one crucible for melting. In the papers by Deb et al. (2003) and Deb and Myburgh (2016) there are two crucibles with capacities W_1 and W_2, respectively; these crucibles can be applied m_1 and m_2 times a day, respectively. We can formulate the CG-ILP for this problem again by using heat patterns, where we distinguish between heat patterns meant for crucible 1 and for crucible 2: in the first (second) type the total amount of molten material required is at most equal to W_1 (W_2) units. We introduce variables x_s and y_s for the two types of heat patterns that measure the number of times that the respective heat patterns are used.

Assuming that we want to minimize the number of days that we need to satisfy all orders, we get the ILP below. Here we use T to denote the number of days that are required; S_1 and S_2 denote the set of heat patterns of type 1 and 2, respectively; and a_{sj} and b_{sj} denote the number of copies of object j ($j = 1, \ldots, n$) that are cast using heat pattern s of type 1 and 2, respectively.

$$\min\ T \qquad \text{subject to} \tag{13}$$

$$\sum_{s \in S_1} x_s \leq m_1 T \tag{14}$$

$$\sum_{s \in S_2} y_s \leq m_2 T \tag{15}$$

$$\sum_{s \in S_1} a_{sj} x_s + \sum_{s \in S_2} b_{sj} y_s \geq r_j \qquad \forall j = 1, \ldots, n, \tag{16}$$

where we further request that T is non-negative and that all x_s and y_s variables are non-negative and integral.

Again, we can apply Column Generation to solve the LP-relaxation, which we can then turn into a near-optimal integral solution by applying some rounding strategy. Suppose that we have solved the LP for a given subset S_1' (S_2') of heat patterns for crucible 1 (2); this yields the shadow prices α and β for Constraints 14 and 15, and π_j for Constraints 16. The reduced cost for the heat pattern for crucible 1 characterized by $(a_1, \ldots, a_n)$ is then equal to

$$-\alpha - \sum_{j=1}^{n} a_j \pi_j.$$

Similarly, the reduced cost for the heat pattern for crucible 2 characterized by $(b_1, \ldots, b_n)$ is then equal to

$$-\beta - \sum_{j=1}^{n} b_j \pi_j.$$

Hence, we must solve two different pricing problems; one for each crucible. Since the only difference between these two pricing problems corresponds to the capacity of the crucible, we get two instances of the KNAPSACK problems that only differ with respect to the volume of the knapsack. Remark that the Dynamic Programming algorithm uses state variables $f_j(t)$ that denote the optimum solution of the instance of the KNAPSACK problem in which we only consider orders $1, \ldots, j$ and in which the size of the knapsack is at most equal to t. The correct values of the state variables $f_j(t)$ are computed recursively, for $j = 1, \ldots, n$ and $t = 0, 1, \ldots, W$, where we use $W = \max\{W_1, W_2\}$. Hence, we can solve these two instances of the KNAPSACK PROBLEM in one run of the Dynamic Programming algorithm, as the desired values are found at $f_n(W_1)$ and $f_n(W_2)$, respectively. After we have solved the LP-relaxation, we can apply the same rounding strategy as before.

If we want to minimize the waste instead of the number of days required, then we can use the objective function

$$\min W_1 \sum_{s \in S_1} x_s + W_2 \sum_{s \in S_2} y_s,$$

which must be minimized subject to Constraint 16 and the constraint that all x_s and y_s are non-negative and integral. This alternative ILP is easily solved using Column Generation again. The same holds if we have more than 2 crucibles; the nice thing here is that we can solve the pricing problems for all crucibles in one run of our Dynamic Programming algorithm, since the corresponding instances of the KNAPSACK problem only differ with respect to the volume of the knapsacks.

4 Restricted availability

Deb et al. (2003) consider the existence of deadlines, which is easily taken care of in their approach, since they know for each heat when it will be produced. In this section we will show that deadlines can easily be included in our ILP formulation, albeit that we need a work-around, since time is not involved in our formulation. In the second part of this section we will consider adding release dates as well, and discuss the adaptations that are required to fit these into the model. We start in our description below with only one crucible; we will later show that our CG-ILP is readily adapted to deal with two (or more) crucibles.

Since the deadlines are known beforehand, we can partition the time into *intervals*, such that the borders of each interval correspond to a deadline (or zero for the first interval); the final interval will run until the end of the time horizon. Let K denote the number of intervals that we obtain in this way. For each interval k ($k = 1, \ldots, K$) we know the orders that can be produced in this interval. Moreover, we know the length of each interval k and hence, given the number of heats per day per crucible, we know the maximum number of heats that can be completed in this interval; we denote this maximum number of heats by h_k, for $k = 1, \ldots, K$.

We include the restricted availability of the orders by using time-dependent heat patterns. For each interval we consider heat patterns that only contain orders that are available in that interval. To distinguish between the intervals, we add to the characterization of each heat pattern $s \in S$ the parameters t_{sk} ($k = 1, \ldots, K$). Here $t_{sk} = 1$ indicates that heat pattern s is meant for interval k; to avoid confusion we assume that each heat pattern is meant for just one single interval. If we want to apply the same heat pattern in two different intervals k and k', then we introduce two heat patterns s and s' for which t_{sk} and $t_{s',k'}$ have value 1, respectively, and all a_{sj} and $a_{s'j}$ are the same for each j. Just like before we introduce a non-negative, integral variable x_s for each heat pattern $s \in S$. Assuming that our goal again is to minimize the total number of heats, then we only have to add the constraints

$$\sum_{s \in S} t_{sk} x_s \leq h_k \quad \forall k = 1, \ldots, K \tag{17}$$

to the original ILP formulation for the basic problem, which existed of the Objective function 5 and the Constraints 6 and 7. When we apply Column Generation to solve the LP-relaxation, then the reduced cost of heat pattern s is equal to

$$1 - \sum_{j=1}^{n} \pi_j a_{sj} - \sum_{k=1}^{K} \lambda_k t_{sk},$$

where λ_k is the shadow price of Constraint 17 for interval k ($k = 1, \ldots, K$). In the corresponding pricing problem we have to minimize the reduced cost over all possible feasible heat patterns. If we consider interval k_0, then the reduced cost becomes

$$1 - \sum_{j=1}^{n} \pi_j a_{sj} - \lambda_{k_0};$$

minimizing this boils down to maximizing $\sum_{j=1}^{n} \pi_j a_{sj}$, where we only allow orders with a deadline greater than or equal to d_{k_0}. We can solve the resulting instances of the KNAPSACK problem again in one run of our dynamic program as follows. We renumber the orders in order of deadline, where order 1 is the one with largest deadline. After renumbering, the set of available orders in interval k becomes equal to $\{1, \ldots, n_k\}$, where n_k is the index of the last job in the list with deadline d_k. Using our Dynamic Programming algorithm we compute the values of the state variables $f_j(t)$, for $j = 1, \ldots, n$ and $t = 0, 1, \ldots, W$. Now we can compute the outcome value of the pricing problem for interval k ($k = 1, \ldots, K$) as $f_{n_k}(W)$, where W is the capacity of the crucible. If there are more,

say C, crucibles, then we still only need one run of the dynamic programming algorithm, since we can find the desired values at $f_{n_k}(W_i)$, for each capacity W_i, with $i = 1, \ldots, C$.

For the column generation to work, we need to provide a feasible starting solution (so a set S of heat patterns for which using only these heat patterns can return a feasible solution). This was straightforward without deadlines, but with deadlines this becomes more challenging. Luckily, we can find such a feasible solution in the same way using column generation again. Hereto, we introduce an extra variable u_j for each order j ($j = 1, \ldots, n$); we let u_j correspond to the number of unfinished copies of order j. We require that each $u_j \geq 0$ and we change Constraint 6, which enforces that we produce at least r_j copies of each order j, to

$$\sum_{s \in S} a_{sj} x_s + u_j \geq r_j$$

Now any set S of heat patterns will provide a feasible solution, although we are of course interested in a solution in which the u_j are all 0. To achieve this, we change the objective to

$$\min \sum_{j=1}^{n} u_j$$

and we solve the resulting LP in the same way as before using column generation. When done it will either return a set of heat patterns S for which this sum is zero, implying that all u_j are zero, or return that this is impossible by finding a minimum value that is positive. In the first case, we can use that set S in the original LP as a feasible starting solution, whereas in the second case, we known that the instance does not have a feasible solution.

After we have solved the LP-relaxation using column generation, we can again construct an integral solution using either rounding up or rounding down. Rounding up, however, can easily violate Constraint 17, which states that we should not use too many heats in a given interval, whereas in case of rounding down, assigning the remaining copies might become infeasible due to Constraint 17 as well, even for instances where the deadlines are not very strict. We can solve this by subtracting some well-chosen Δ from the righthand-side of Constraint 17. When Δ is chosen large enough, rounding up will return a feasible integral solution as well and thus also when rounding down a feasible solution can be found. A basic choice for Δ would be the minimum value such that no matter what fractional solution we encounter, we can always round up and find a feasible integral solution. There are $n+k$ constraints, implying that at most $n+k$ variables x_s will have a positive value in our solution, so choosing $\Delta = n+k$ will certainly work. Note that this value can be rather large, as we assume that all heat patterns will be concentrated in one interval. In practice, one might change Δ iteratively which can be dealt with very easily in the LP. However we did not examine this in our experiments.

When rounding down, we can still use the DM-ILP where we apply binary search on H. The only difficulty is to decide how to assign the additional H heats to the intervals. Luckily as we only have deadlines, we can simply assign the heats to the first possible interval, as all possible heats in a later interval can always be used in an earlier interval. In this way, we can still use binary search on H to determine an optimal way of assigning the remaining copies. The DM+-ILP will of course also work.

If the goal is not to minimize the total number of heats but to finish as early as possible, then we can use as our objective to minimize the number of heat patterns that must be used in interval K, that is,

$$\min \sum_{s \in S} t_{sK} x_s;$$

this change in the objective function is easily dealt with.

If we have more than one, say C, crucibles, then we can apply the same strategy as we used in Section 3. To each heat pattern $s \in S$ we then assign a parameter q_{sc} that gets the value 1 if

heat pattern s is meant for crucible c. We can readily compute the maximum number of heat patterns that can be applied for crucible c ($c = 1, \ldots, C$) in period k ($k = 1, \ldots, K$), and add the corresponding constraints to the LP. The pricing problem corresponds then to finding the heat pattern with minimum reduced cost for each interval and each crucible; we can find the best heat pattern for all situations by one run of our Dynamic Programming algorithm.

Our final extension concerns the addition of release dates, which decree that the order cannot be cast before the release date; one possible reason for having a release date may be that we need some special equipment for casting that is not standard available. The presence of release dates next to deadlines complicates the problem from a computational point of view. We can still work with intervals, the borders of which now correspond to either a release date or a deadline. When we solve the LP-relaxation, then we have to solve the pricing problem, which depends on the interval as before. Unlike before, we must now run our Dynamic Programming algorithm for each interval separately instead of using only one run, since the set of available orders can vary arbitrarily per interval. This slows down solving the LP-relaxation drastically, as is shown in our computational experiments. Once we have solved the LP-relaxation we can use the same rounding strategy as for the case in which we only have deadlines. However when rounding down, we can not simply use binary search on H any more, as it is unclear how to assign the heats to the intervals. We can still use the DM+-ILP, which is what we did in our experiments.

5 Computational experiments

In this section, we perform some computational experiments to test the CG-ILP. For consistency, we always start with the version of CG-ILP with the u_j variables to find a feasible solution, even though this is not required for instances without deadlines or release dates. Since CG-ILP returns nearly optimal results, we mainly focus on the computation time. In each experiment, we run our algorithm 10 times independently and report the average running time needed. We try both rounding the solution up and rounding the solution down where, after rounding down, we always use the DM+-ILP to assign the remaining items. We chose this option over the binary search on H on DM-ILP as it also works with release dates, so it allows us to use the same strategy all the time for consistency. We also report the number of columns (or heat patterns) that have been generated when solving the LP-relaxation as this is an important metric for the computation time needed and also illustrates how many heat patterns are actually explored. All experiments were performed on a computer with an Intel I7 10th generation CPU. For solving the ILP, we used Gurobi 11.0.2.

We will start by some basic instances from Deb and Myburgh (2016) as a proof of concept. Afterwards, we focus on the relation between the request values r_j ($j = 1, \ldots, n$) and the computation time and show that unlike in the Genetic Algorithm or DM-ILP, the request values matter very little for the computation time. Next we focus on the number of orders n to be molded. At the end, we also consider release dates and deadlines: we assume that there are a few “priority orders" that have to be molded in a very short interval. We test how this impacts the CG-ILP. Lastly, we increase the capacities of the crucibles: this should make the pricing problem harder to solve as the DP will take more time to solve, so we will test how much this impacts the computation time. At the same time, this will also give insight in whether solving the pricing problem or maintaining the LP is the bottleneck in our algorithm.

We start with the very basic example from Deb and Myburgh (2016), which can be found in their Table 1: this instance contains 10 orders in which a total of 200 copies are requested. There is only one crucible used per day with a capacity of 650 kg. Deb and Myburgh (2016) show that the optimal number of heats is 31 in this instance. Both the DM-ILP and the Genetic Algorithm can solve the instance optimally in around 0.05 seconds. Our CG-ILP formulation took 0.01 ± 0.001 seconds, which is faster and generated only 21 columns. However, it does not solve the problem optimally due to rounding and instead returns a solution with 34 heats when we round up the fractional values. If we round down instead of rounding up and place the remaining copies optimally

using the DM+-ILP, we need 0.017 ± 0.002 seconds and we also find the optimal solution of 31 heats.

Deb and Myburgh (2016) also explore as a proof of concept what happens when this simple instance is scaled up: there are still 10 orders, but instead of 20 copies requested per order, all orders request 130 copies (apart from the first order, which requests 127 copies). In this case, the optimal number of heats is 200. However the DM-ILP was unable to find the optimum even when it ran for 15 hours. The Genetic Algorithm still performed very well and solved the problem optimally in only 0.19 seconds. The CG-ILP solved the problem in only 0.012 ± 0.001 seconds and again generated 21 columns. However it returned a solution with 203 heats rather than the optimal 200 heats due to the rounding up. Again, when we round down instead of rounding up, we find the optimal solution with 200 heats in 0.016 ± 0.002 seconds. This very simple instance already illustrates some key properties of the CG-ILP algorithm: it is very scalable or even independent from the number of requested copies and it will return a solution which has generally a few heats more than the optimum when we round up. When we round down, it generally finds a slightly better solution (and hopefully even an optimal one even though this cannot be guaranteed), but that requires a little more time.

We will now examine a larger instance from Deb and Myburgh (2016), which can be found in their Table 2: this instance contains 10 orders with a total of $550,666$ requested copies and a total weight of $56,352,140$ kg. We have two separate crucibles; the first one has a capacity of 650 kg, which is used 10 times a day, whereas the second one has a capacity of 500 kg, which is used 13 times per day. If we assume an integral number of days has to be used, we need at least 4337 days to melt all the orders, which corresponds to $99,751$ heats. Unlike with the previous examples, Deb and Myburgh (2016) do not aim to solve this problem to optimality, but only want to find a solution that uses at most $100,000$ heats. Their Genetic Algorithm succeeds at this in approximately 5 minutes. In contrast, CG-ILP still only requires 0.01 ± 0.001 seconds to solve this problem, and it also returns a solution with the optimal number of 4337 days with heats. It again only generates 21 columns. Also when we round down instead of up and solve the problem for the remaining copies optimally, we only need 0.013 ± 0.002 seconds (and we also find the optimum).

Finally, Deb and Myburgh (2016) scaled this large instance up by a factor 1000 to achieve a "billion-variable problem". Their Genetic Algorithm aimed to find a solution with at most $100,000,000$ heats and it succeeded at this task. However, this took approximately $535,503$ seconds which is over 6 days. We also tested CG-ILP on this very large instance. It needed 0.011 ± 0.001 seconds again and solved the problem optimally by returning a solution with the minimum number of $4,336,319$ days. Again only 21 columns are generated. Also when rounding down instead of up, we only need 0.013 ± 0.002 seconds again.

We have tested CG-ILP even further and test the instance from Deb and Myburgh (2016) presented in their Table 2 scaled up with factors $1, 10, 100, 1000, 10^4, 10^5, 10^6, 10^7, 10^8, 10^9$. Note that the instance scaled up with factor 1000 is the largest instance examined in Deb and Myburgh (2016), which contained one billion variables. In our Table 1 below, we show the computation time needed for all instances. All instances are solved (almost) optimally: for the factor 100 and the factor 10^8 instance we achieve a value of 1 above the optimum when rounding up and an optimal solution when rounding down. All other instances are guaranteed to be solved to optimality as with fewer days, not enough metal could possibly be molten to meet the required weight. We see that all instances require about the same running time, even though the number of requested copies drastically increases. Also the number of columns generated is the same (21) for all instances. This clearly illustrates that the CG-ILP formulation is not really influenced by the number of requested copies per order.

Scale-Up Factor	Rounding Up	Rounding Down	Columns Generated
1	0.011 ± 0.001	0.013 ± 0.002	21
10	0.012 ± 0.001	0.017 ± 0.003	21
100	0.01 ± 0.001	0.02 ± 0.002	21
1000	0.011 ± 0.001	0.014 ± 0.003	21
10^4	0.01 ± 0.001	0.018 ± 0.003	21
10^5	0.011 ± 0.001	0.018 ± 0.005	21
10^6	0.01 ± 0.001	0.02 ± 0.003	21
10^7	0.011 ± 0.001	0.016 ± 0.003	21
10^8	0.011 ± 0.001	0.019 ± 0.003	21
10^9	0.011 ± 0.001	0.015 ± 0.002	21

Table 1: The computation time needed in our scale-up study. All times are in seconds.

Below we tested the impact of the number of orders n. To do this, we created instances that had the same crucibles as the large instances from Deb and Myburgh (2016): one crucible of weight 500 kg which is used 13 times per day and one crucible of weight 650 kg which is used 10 times per day. We add n orders and for each order we draw its weight uniformly from $[100, 650]$ and its number of requested copies uniformly from $[1, 1000]$. We tried the values $10, 20, 50, 100, 200, 500, 1000$ for n. In our Table 2 below, we show the computation time in seconds that CG-ILP needs, both when the fractional solution is rounded up and down. Remark here that the difference in computation time is solely due to running the DM+-ILP to assign the lacking copies of the objects to additional heats. We see that having more orders clearly increases the computation time. This is also easily explainable as the number of columns generated increases too. We also see that rounding down takes slightly longer than rounding up, just as expected as we need to solve another ILP afterwards. The difference however is not very large, even though in theory the DM+-ILP becomes extremely hard to solve for large n. This is probably mainly because there are far fewer requested copies in our case as the large majority of copies have already been molded.

n	Rounding Up	Rounding Down	Columns Generated
10	0.01 ± 0.001	0.017 ± 0.007	18.4 ± 2.109
20	0.019 ± 0.005	0.028 ± 0.013	40.5 ± 3.456
50	0.042 ± 0.01	0.045 ± 0.016	100.2 ± 10.679
100	0.255 ± 0.027	0.307 ± 0.025	226.9 ± 22.928
200	0.972 ± 0.064	1.16 ± 0.102	469.6 ± 30.361
500	8.192 ± 0.935	9.313 ± 1.075	1535.4 ± 169.915
1000	31.806 ± 10.634	35.568 ± 12.516	3478.8 ± 252.719

Table 2: The effect of the number of orders on the computation times. All times are in seconds.

Next, we added deadlines. We have n orders with random weights from $[100, 500]$ and a random number of requested copies from $[100000, 200000]$. We again have two crucibles: one of size 650 kg and one of size 500 kg that are used 10 and 13 times per day, respectively. We further add a deadline to all orders. We do this in such a way that until the deadline of order 1, we have enough capacity to melt 3 times the needed weight of metal for all copies of order 1. In the same way, we pick deadline 2 such that we have exactly enough capacity until deadline 2 to melt 3 times the needed weight of metal for all copies of orders 1 and 2, and so on for all deadlines. Note that we need the rather large number of requested copies as otherwise the impact of Δ on the number of available days becomes too large and some instances might become infeasible. In practice, a lower

Δ could be used to solve this problem, but we did not examine that option here. In our Table 3 below, we show the computation time needed by CG-ILP for these instances. For n we tried $10, 20, 50, 100, 200, 500, 1000$ just as before. We also show the result both when we round up and round down. We see just like in our Table 2 that the computation time highly depends on the number of orders n. The increase in computation time is not that high, probably because we can very efficiently solve the pricing problem when we only have deadlines.

n	Rounding Up	Rounding Down	Columns Generated
10	0.023 ± 0.004	0.033 ± 0.006	38 ± 3.629
20	0.039 ± 0.007	0.076 ± 0.014	88.7 ± 9.977
50	0.191 ± 0.016	0.401 ± 0.033	240.4 ± 14.546
100	0.899 ± 0.08	1.748 ± 0.144	537.5 ± 23.916
200	4.552 ± 0.194	8.996 ± 0.369	1126.9 ± 27.048
500	54.413 ± 6.521	88.595 ± 6.712	3012.2 ± 172.46
1000	508.424 ± 29.748	654.367 ± 73.754	6539.9 ± 98.301

Table 3: The computation time needed by our improved ILP when we add deadlines to the n orders. All times are in seconds.

Next, we tested our algorithm on a set of instances to which we added both deadlines and release dates. Just as before, there are n orders with random weights from $[100, 500]$ and a random number of requested copies from $[100000, 200000]$; for melting we have two crucibles of size 650 kg and 500 kg. However, in this case, half of the orders has neither a deadline nor release date, whereas the other half has a narrow interval in between it must be cast. The interval is exactly long enough to melt 3 times the required weight in metal, where the start time of the interval is chosen randomly from $[1, 100000 \cdot n]$. We tried the values $10, 20, 50, 100$ for n, since for this type of problems with release dates, CG-ILP requires considerably more time. In our Table 4, we tabulated the needed computation time and the number of generated columns. Just as before, we see that the computation time depends greatly on the number of orders n; for this type of problems with both release dates and deadlines, the computation time increases even faster when n increases. However we see that the number of generated columns does not increase very fast and is comparable to our Table 3 with only deadlines. The difference between computation time and number of generated columns is probably mainly caused by the pricing problem becoming harder to solve: now we have $O(n)$ intervals and for all of them, we have to solve the resulting instance of the KNAPSACK problem separately, so the computation time for solving the pricing problem increases by a factor n.

n	Rounding Up	Rounding Down	Columns Generated
10	0.053 ± 0.006	0.065 ± 0.006	42.5 ± 4.447
20	0.311 ± 0.028	0.35 ± 0.023	97.7 ± 7.281
50	4.806 ± 0.523	5.074 ± 0.579	267.9 ± 19.898
100	40.752 ± 3.309	41.277 ± 3.398	557.6 ± 41.468

Table 4: The computation time needed by our improved ILP when we both have deadlines and release dates. All times are in seconds.

Finallly, we increased the capacities of the crucibles to test how much this impacts the running time of CG-ILP. For our set of orders we use the large instance from Deb and Myburgh (2016)

as presented in their Table 2; this is the instance with scale-up factor 1 in our Table 1. We also use the same two crucibles, but we scale up their capacities with factors $1, 2, 5, 10, 20, 50, 100$. We again tried both rounding up and rounding down and recorded the number of columns needed. In our Table 5 below, the results can be found. All instances are solved optimally, except factors 20 and 50 for which rounding up returns a solution with one day more than the optimum; the optimum is found when we round down, though. We see that, as expected, the computation time increases, although especially for low factors, this increase is rather slow. This is probably caused by the fact that for small weights, solving the pricing problem can be done very fast and most time is spent on the LP itself. Also we see that the number of generated columns decreases when the weights increase, which also makes the problem easier to solve. From around factor 10, we see an almost linear increase since from then onwards, which indicates that the computation time is mainly spent on solving the pricing problem. Still the problem remains solvable ihttps://www.uu.nl/en/events/beyond-technology-utrecht-ai-eventn less than 1 second, even when the crucibles have a high capacity.

Scale-Up Factor	Rounding Up	Rounding Down	Columns Generated
1	0.013 ± 0.004	0.015 ± 0.004	21
2	0.015 ± 0.002	0.017 ± 0.002	19
5	0.022 ± 0.002	0.026 ± 0.002	18
10	0.033 ± 0.002	0.037 ± 0.004	17
20	0.057 ± 0.006	0.065 ± 0.006	17
50	0.123 ± 0.009	0.128 ± 0.008	17
100	0.245 ± 0.009	0.253 ± 0.012	17

Table 5: The impact of increasing the size of the crucibles on the computation time of our improved ILP. All times are in seconds.

6 Conclusion

In this paper we have shown how the casting problem can be formulated using an ILP formulation based on so-called *heat patterns*, which specify the number of copies for each order that can be cast using the iron that is melted during one heat. For this ILP formulation we find a solution by solving the LP-relaxation using column generation followed by rounding; this solution provably uses a number of heats that is very close to the theoretical minimum number of heats required. Since this can be done within a second for the instances presented by Deb and Myburgh (2016) instead of using a week to show existence of a worse solution, we have clearly refuted the claim by Deb and Myburgh (2016) that instances of real-life size cannot be solved by ILP. The main improvement of our ILP is that it treats the number of times a heat pattern is used as a variable, instead of a parameter that impacts the size of the ILP. As a result working with an extremely large number of copies per order does not make the slightest difference. The number of orders does impact the size of the ILP and thus the computation time, but still instances with up to 1000 orders can be solved in half a minute. A similar remark holds for the capacity of the crucibles; problem instances with capacities that are scaled up by a factor 100 can still easily be solved within a second. Finally, we have shown how our approach can deal with deadlines and eventually release dates. Adding release dates next to deadlines makes the LP-relaxation much harder to solve, since the pricing problem becomes much harder to solve. Still, we can solve instances with up to 100 orders in less than a minute.

References

1. K. Deb, A.R. Reddy, and G. Singh (2003). Optimal scheduling of casting sequence using genetic

algorithms. *Materials and Manufacturing Processes 18*, pp. 409–432.
https://doi.org/10.1081/AMP-120022019

2. K. Deb and C. Myburgh (2016). Breaking the billion-variable barrier in real-world optimization using a customized evolutionary algorithm. *GECCO '16: Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pp. 653–660.
https://doi.org/10.1145/2908812.2908952
3. P.C. Gilmore and R.E. Gomory (1961). A linear programming approach to the cutting-stock problem. *Operations Research 9*, pp. 849–859.
https://pubsonline.informs.org/doi/10.1287/opre.9.6.849
4. P.C. Gilmore and R.E. Gomory (1963). A linear programming approach to the cutting-stock problem - Part II. *Operations Research 11*, pp. 863–888.
https://www.jstor.org/stable/167827
5. P.C. Gilmore and R.E. Gomory (1965). Multistage cutting stock problems of two and more dimensions. *Operations Research 13*, pp. 94–120.
https://www.jstor.org/stable/167956
6. L.V. Kantorovich (1960). Mathematical methods of organizing and planning production. *Management Science 6*, pp. 366–422.
https://www.jstor.org/stable/2627082
7. H. Kellerer, U. Pferschy, and D. Pisinger (2004). *Knapsack problems*, Springer-Verlag, Berling-Heidelberg.