TPLP **23** (4): 848–864, 2023.
doi:10.1017/S147106842300025X First published online 21 July 2023

Logic-Based Benders Decomposition in Answer Set Programming for Chronic Outpatients Scheduling

PAOLA CAPPANERA
DINFO, Università degli Studi di Firenze, Italy
(*e-mail:* paola.cappanera@unifi.it)

MARCO GAVANELLI and MADDALENA NONATO
DE, Università degli Studi di Ferrara, Italy
(*e-mails:* marco.gavanelli@unife.it, maddalena.nonato@unife.it)

MARCO ROMA
DINFO, Università degli Studi di Firenze, Italy
(*e-mail:* marco.roma@unifi.it)

submitted 30 May 2023; revised 21 June 2023; accepted 22 June 2023

Abstract

In answer set programming (ASP), the user can define declaratively a problem and solve it with efficient solvers; practical applications of ASP are countless and several constraint problems have been successfully solved with ASP. On the other hand, solution time usually grows in a superlinear way (often, exponential) with respect to the size of the instance, which is impractical for large instances. A widely used approach is to split the optimization problem into subproblems (SPs) that are solved in sequence, some committing to the values assigned by others, and reconstructing a valid assignment for the whole problem by juxtaposing the solutions of the single SPs. On the one hand, this approach is much faster due to the superlinear behavior; on the other hand, it does not provide any guarantee of optimality: committing to the assignment of one SP can rule out the optimal solution from the search space. In other research areas, logic-Based Benders decomposition (LBBD) proved effective; in LBBD, the problem is decomposed into a master problem (MP) and one or several SPs. The solution of the MP is passed to the SPs that can possibly fail. In case of failure, a no-good is returned to the MP that is solved again with the addition of the new constraint. The solution process is iterated until a valid solution is obtained for all the SPs or the MP is proven infeasible. The obtained solution is provably optimal under very mild conditions. In this paper, we apply for the first time LBBD to ASP, exploiting an application in health care as case study. Experimental results show the effectiveness of the approach. We believe that the availability of LBBD can further increase the practical applicability of ASP technologies.

KEYWORDS: answer set programming, logic-based Benders decomposition, outpatients appointment scheduling, chronic patients with comorbidities

1 Introduction

Answer set programming (ASP) is recently gaining momentum not only in the logic programming area but also in the constraint optimization and Operations Research communities. ASP relies on the Stable Model Semantics Gelfond and Lifschitz (1988);

in an ASP formulation of a combinatorial problem, the solution is encoded in such a way that a stable model (or answer set) of the program corresponds to a solution of the problem. Most ASP solvers divide the solution process into two steps: a grounding phase, in which a ground program having the same stable models of the original program is generated, followed by a solving phase in which the answer sets are computed. The number of applications is impressive, deserving five surveys in a few years Erdem *et al.* (2016); Dal Palù *et al.* (2018); Falkner *et al.* (2018); Kifer and Liu (2018); Schüller (2018).

However, many problems coming from real-life applications cannot be solved in reasonable time because their solution time is superlinear (often, exponential) with the instance size and the instances are very large. As splitting large problems can simplify them, the ASP literature reports many applications in which a difficult problem is split into two (or more) subproblems (SPs) that are then solved in sequence (see e.g., the studies presented by Guido *et al.* (2020), Cardellini *et al.* (2021), Caruso *et al.* (2023), and El-Kholany *et al.* (2022) just to name a few). Such an approach could be described as follows. Consider a problem P, in which a function $f(x, y)$ is maximized subject to a set of conditions $C(x, y), C_x(x), C_y(y)$, where x and y are two vectors of variables, that range respectively in the domains D_x and D_y:

$$P: \qquad \max\{f(x,y) \mid x \in D_x, y \in D_y, C_x(x), C_y(y), C(x,y)\}. \tag{1}$$

Suppose that solving P is too computationally demanding. The splitting approach would split P into, for example, two SPs P_x and P_y (we simplify the exposition by considering two SPs, although the approach could be extended to more levels and SPs), in which P_x might be responsible of assigning values to x variables, while P_y to the y variables. One could then find the optimal assignment x^* for P_x:

$$x^* = \arg\max_x\{f(x,y) \mid x \in D_x, C_x(x)\}, \tag{2}$$

then solve the remaining SPs:

$$y^* = \arg\max_y\{f(x^*,y) \mid y \in D_y, C_y(y), C(x^*,y)\}, \tag{3}$$

and finally provide the pair (x^*, y^*) as proposed solution of the whole problem (1).

This approach can be considerably faster than solving the whole problem (1) due to the superlinear solving time; on the other hand, it also has a number of issues. First, the optimal solution x^* for the first SP might be impossible to extend to the y variables, that is, there might be no assignment to the y variables such that $C(x^*, y)$ is satisfied. In some applications, one might be able to split the problem in such a way that for each value of the x variables there always exists an assignment to the y variables (and, indeed, in the aforementioned applications Guido *et al.* (2020); Cardellini *et al.* (2021); Caruso *et al.* (2023); El-Kholany *et al.* (2022) the authors were able to find such an intelligent splitting); nevertheless, this limits the applicability of the splitting approach only to some specific applications. Second, even if for x^* there exists an assignment y^* that satisfies all constraints $C(x^*, y^*)$, the pair (x^*, y^*) might be not optimal for the global problem P. In general, committing too early to the solution of one SP might prevent the optimal solution to be found. This splitting approach could be thought of as a (very clever) greedy algorithm, in which one solves to optimality the first SP, greedily commits to it, and then solves the second SP.

Benders decomposition is a technique to decompose a problem into SPs while retaining the ability to obtain the optimal solution and prove its optimality. It was born in the realm of Operations Research and relies on the duality theory of Linear Programming. It was later extended to approaches that cannot rely on a duality theory in the so-called logic-based Benders decomposition (LBBD) Hooker and Ottosson (2003).

In LBBD, a master problem (MP) is solved first and provides tentative values that are passed to the SPs. Consider the generic problem in equation (1); the optimal solution x^* of the master problem (MP) is obtained as in equation (2), and it is provided to the SP.

Now, the SP (equation (3)) is solved using the suggested values x^*. Two situations may occur: either the SP is proven infeasible or its optimal solution y^* is found.

In case of infeasibility, clearly the assignment x^* is not acceptable for the whole problem (1), so a constraint that rules out x^* is forwarded from SP to MP. Such a constraint is named a *feasibility cut*, and it is added to the MP formulation.

In case SP is feasible, its optimal solution y^* is obtained together with the corresponding value of objective function $v^* = f(x^*, y^*)$; again a new constraint, called *Benders cut*, is returned to the MP. Such constraint relates the MP variables x with v^* through a function $B_{x^*}(x)$, imposing an upper bound on the value of the objective function:

$$B_{x^*}(x) \leq v^*. \tag{4}$$

How to formulate the function $B_{x^*}(x)$ is left to the designer of the LBBD solution process, and it is a challenge, as it can highly influence the efficiency of the decomposition. In order to devise Benders cuts, the idea is that the SP solver proved that v^* was optimal for the SP, obtaining a proof of optimality that can be written as $(\forall y)\ f(x^*, y) \leq v^*$; such a proof of optimality might also be extended to include values of x different from x^*. Delving into further details of this fascinating subject would distract us from the main topic of this paper; the interested reader can refer to Hooker (2019) for an introduction to LBBD and some examples of $B_{x^*}(x)$ functions in practical instances.

In both cases (SP feasible / infeasible), a new iteration is started: the MP with the additional constraints is solved again, and the iteration continues until either the MP is proven infeasible (and in such a case, the whole problem is infeasible) or the optimal solution is found. In any iteration, the optimal solution of the MP is an upper bound of the whole problem (1): since the MP contains a subset of the constraints of the whole problem P, it is actually one of its relaxations, so its optimal value is optimistic with respect to the real optimum of (1). Again, in each iteration, the pair (x^*, y^*), obtained by juxtaposing the optimal solution y^* of the SP (3) with the optimal solution x^* of the MP, is a valid solution, so its value $f(x^*, y^*)$ is a lower bound of the whole problem P. If at any iteration the lower bound is equal to the upper bound, then (x^*, y^*) is provably optimal for problem P. A sufficient condition for the termination of the iteration is that the bounds (4) are valid and the variables' domains are finite Hooker and Ottosson (2003).

In various interesting cases, this procedure can be simplified; in particular if the SPs are feasibility problems, that is, in equation (1) the objective function f does not depend on the y variables, the LBBD algorithm can be described as in Algorithm 1.

Algorithm 1 LBBD scheme in case the subproblem is a feasibility problem.

```
$i \leftarrow 0$
repeat
    $i \leftarrow i + 1$
    $x_i^* = \text{Solve}(MP)$
    if MP is infeasible then return infeasible
    $y_i^* = \text{Solve}(SP, x_i^*)$
    if SP is infeasible then generate a feasibility cut ruling out $x_i^*$ and add it to $MP$
until SP is feasible
return $(x_i^*, y_i^*)$
```

In the rest of the paper, we will focus on decompositions in which the SPs are feasibility problems, that is, the y variables do not explicitly occur in the objective function.

LBBD can provide strong speedups, in particular when there is a hierarchical relation between the solution of the MP and that of the SP, so that, once the assignment is found for the MP, the SP becomes easy in some sense (it could be a theoretically easy problem – for example, a problem in P – or a problem that is experimentally found to be relatively easy). Further speedups can be obtained when once a solution for the MP is found, the rest of the problem P consists of independent SPs that can be solved independently (even in parallel).

In this work, we use for the first time LBBD in an ASP-based solving scheme. We show its application on a challenging real-life problem coming from the healthcare domain.

2 Case study

Cappanera *et al.* (2022) (2023) addressed a scheduling problem involving chronic patients with comorbidities. Many patients suffer from so-called non-communicable chronic diseases (NCDs), such as diabetes, hypertension, cirrhosis, obesity, and so on. For most NCDs, there exist well-assessed medical guidelines involving periodic health services to be delivered at hospital premises – think of dialysis for patients with renal failure. Most patients are not hospitalized but access hospital premises as outpatients; many of them have more than one NCD (comorbidity). Patients are assigned personalized care plans, that is, clinical pathways (CPs), that merge the medical guidelines of all diagnosed NCDs, customized to the specific patient. A CP's health services are known *a priori* over a mid term horizon, which allows for well in advance planning. Scheduling the health services of a CP means to assign a date, a time, and an operator to each service the patient must receive at the hospital. Such process can be challenging because appointment dates must comply as much as possible with the ideal frequency and other time constraints due to interference (a treatment may alter the result of an exam taken after it) or precedence (a consultancy requires recently taken exams). Finally, if there is not enough availability within the public hospital, a service can be provided by private health services at a higher cost for the National Health Service. The centralized management of the CPs of all patients would optimize the usage of public resources and ensure fairness. This yields a very challenging problem that we call *NCD Agenda problem*.

3 Related works

Several healthcare problems have been tackled with ASP (see the review by Alviano *et al.* (2020)). We recall the most notable contributions highlighting decomposition, if any. Caruso *et al.* (2023) schedule preoperative exams for outpatients by dividing the problem in two steps: first, exam areas are staffed and patients are given an appointment day; then, exams starting times are set, complying with first-level decisions, maximizing the served patients and minimizing waiting time. Each phase is executed once, with no feedback; to ensure feasibility in phase two, demand is overestimated in phase one.

A schedule for multiple appointments for rheumatic outpatients at a Hospital Day Service is presented by Guido *et al.* (2020). Patients are partitioned into three classes with decreasing priority; to reduce computing time the schedule is computed separately for each priority class.

Nurse (re)scheduling was addressed by Alviano *et al.* (2018), improving on the representation of hospital and work balance constraints presented by Dodaro and Maratea (2017). Nurse scheduling consists of determining a shift assignment for each nurse for a given planning horizon such that working hours, shift mix, and rest days comply with hospital rules. Rescheduling is due in case of nurse temporary absences and consists of feasibly scheduling vacant duties, while minimizing deviations from the original schedule.

Appointment scheduling for chemotherapy treatments must deal with the availability of special equipment that is assigned to a patient for the whole session Dodaro *et al.* (2021). A treatment encompasses up to four subsequent steps, some of which are optional, whose duration is patient dependent and known. In case of multiple treatments, a treatment frequency is given. A weekly problem is solved, as well as a rescheduling one.

Rehabilitation sessions for inpatients are scheduled by Cardellini *et al.* (2021). Two types of resources are present: gyms and operators. Solution quality criteria and constraints include continuity of care and preferred time slots on the patient side, and workload balancing and abiding by working rules on the operator side. The daily problem is decomposed into two subsequent decision phases. In the first, the board, patients are assigned to operators; in the second, the agenda, a starting and ending time is set for each session according to the board. As there is no feedback, there is no guarantee that a feasible board-compatible agenda exists. To this aim, potential overlapping are admitted: some sessions are partially turned from one-to-one care to supervision (one operator supervises a few patients at a time).

Finally, ASP has been proved effective in the (re)scheduling of operating rooms (OR). A planned surgery requires a free bed at the specialty ward or at intensive care units, starting from surgery date for the predicted length of stay Dodaro *et al.* (2022), and a bed at the post-anesthetic care unit for temporary post-surgery staying Galatà *et al.* (2021). Since a surgical team is made of surgeons, anesthesiologists, and nurses, the whole surgery slot must be fully contained into the current working shift of each team member. Based on its specialty, priority, special needs, and expected duration, a request is assigned a day and a time during the OR time blocks reserved to its specialty.

Out of the healthcare domain, El-Kholany *et al.* (2022) improve on a previous study and present a decomposition scheme in ASP for Job Shop Scheduling, driven by a machine learning algorithm. There is no feedback from the SPs to the MP, each SP is solved only once, and thus the resulting algorithm cannot prove optimality of the found solution (it is a heuristic algorithm).

In conclusion, we observe that ASP proved to be able to capture and easily represent the complex features of several challenging problems. Decomposition schemes are often implemented, motivated by the need for solving large instances in a reasonable time; however, they are implemented in such a way that the optimal solution could be overlooked and optimality cannot be guaranteed. To the best of our knowledge, we are the first to propose the use of LBBD in ASP. Out of the ASP area, LBBD has been successfully applied in Integer Linear Programming Riise *et al.* (2016) and Constraint Programming frameworks Zhu *et al.* (2023), often as an hybrid algorithm. Fazel-Zarandi and Beck (2009) use a hybrid CP-MILP approach for facility location, Benini *et al.* (2011) for resource allocation for multicore platforms, just to name a few. Two major problems in manufacturing – usually solved in pipe, to the detriment of optimality – were handled together by Zhu *et al.* (2023), exploiting LBBD and a clever CP-based formulation. A comprehensive survey was published by Hooker (2019). LBBD was also applied to a railway timetabling problem using Satisfiability Modulo Theory Leutwiler and Corman (2022).

Finally, the splitting set theorem Lifschitz and Turner (1994) provides syntactic conditions under which the stable models of a program can be obtained extending the stable models of one subprogram. LBBD is applied on a different level: the level of modeling an optimization problem and decomposing it into SPs, even if the splitting set theorem could be exploited to have synergies with LBBD.

4 NCD agenda formalization

Let us consider a planning horizon (set of available days) $\mathbf{H} = \{1, \ldots, h\}$, a set of health services $\mathbf{S}$, and a set of patients $\mathbf{P} = \{p_1, \ldots, p_{N_P}\}$. For each patient p, a CP is known, consisting of a set of *packets*. Each packet π is a set of services to be delivered on the same date, even if they are provided by different care units.

The appointment dates of each CP should satisfy the following *CP constraints*:

Frequency: Often a pathway contains sets of packets corresponding to recurring services; for each packet there is an ideal date (ensuring that the patient is serviced with the correct frequency) and the packet should be scheduled within a tolerance from the ideal date. The tolerance depends on the pathway, and it is such that the tolerance windows of consecutive occurrences of the same packet are disjoint.

Interdiction: if s_i interdicts service s_j for δ days and s_i is scheduled in $\tau(s_i)$, then $\tau(s_j)$, the appointment date of s_j, is such that $\tau(s_j) \notin [\tau(s_i), \tau(s_i) + \delta]$. Interdiction constraints are always satisfied if one of the two services is not scheduled.

Necessity: if s_i requires s_j, an interval $[\delta_{min}, \delta_{max}]$ is provided; service s_j should be scheduled on day $\tau(s_j) \in [\tau(s_i) + \delta_{min}, \tau(s_i) + \delta_{max}]$ and cannot be scheduled in the (right-open) interval $[\tau(s_i), \tau(s_i) + \delta_{min})$.

A second class of constraints concerns resource assignment: each service s has a service type and a duration; each scheduled service should be assigned to an operator of the care unit that provides that service type. Each operator at the care unit has a working shift (start and end time, potentially empty) for each day in the horizon. The following *daily agendas constraints* hold: (i) all services provided by an operator should be completed within the operator shift and (ii) without overlaps *(no patient overlapping)*, (iii) each patient cannot receive two services in parallel *(no service overlapping)*, (iv) a service cannot be interrupted and resumed at a later time *(no preemption)*, as well as (v) a scheduled service is delivered by a single operator *(no split-service)*.

A feasible schedule assigns an appointment date $\tau(\pi)$ to each scheduled packet π, as well as a time and an operator to the services of π so that all constraints are satisfied. If a packet is not scheduled, the patient will receive the same services from a private clinic at a higher cost. The objective is to maximize the number of scheduled packets.

For example, in Figure 1, the set of patients is $\mathbf{P} = \{p1, p2\}$; the pathway of $p1$ is made of just one packet π_1 which includes two services; the color (red or blue) represents the service type and each service is associated with the care unit of the same color. Care unit 1 (red) has four time slots of availability on day 1, three slots on day 2, and two slots on day 3. Note also the different start times of the operators' shifts: the operator of care unit 1 on day 3 starts earlier than that of care unit 2.

5 ASP approaches

ASP relies on the Stable Model Semantics Gelfond and Lifschitz (1988). An ASP program is a set of clauses $\mathtt{h{:}\text{-}b_1, \ldots, b_n}$, where $\mathtt{h}$ is an atom $\mathtt{p(t_1, \ldots, t_m)}$ or a choice $\{\mathtt{p(t_1, \ldots, t_m)}\}$ and $\mathtt{b_i}$ can either be literals of the form $[\mathtt{not}]\mathtt{p(t_1, \ldots, t_k)}$, possibly followed by a condition $: \mathtt{c_1, \ldots, c_k}$, or an aggregate $\mathtt{\#sum}\{\mathtt{t_1, \ldots, t_m : c_1, \ldots, c_k}\} \circ \mathtt{n}$ where $\circ$ is a comparison operator $<, =, >=, \ldots$. A clause without head is called an integrity constraint (IC), and its body must evaluate to false in every Stable Model (or Answer Set) of the program. Optimization components can be added by means of weak ICs, with syntax $: \sim$*body*; the aim will be to find an answer set that satisfies all ICs while satisfying the maximum number of weak ICs. For the full ASP syntax, see Calimeri *et al.* (2020).

We recap the ASP formalization of the agenda component of the NCD Agenda Cappanera *et al.* (2022) in Section 5.1, and the scheduling of services within the day in Section 5.2. The LBBD approach is developed in Section 5.3.

5.1 Scheduling services – Date assignment

The input data are provided by the following predicates:

- `occurrence_to_schedule(Patient,Packet)` provides the packets that should ideally be scheduled for each patient; the ideal date is `ideal_date(Patient, Packet, IdealDate)`, but a tolerance is accepted; predicate `within_tol(Packet, Day, IdealDate)` checks if `Day` stands within the tolerance.
- The set of available days for the scheduling is provided by `day(D)`;
- `service_in_packet(Srv,Pck)` means that service `Srv` belongs to packet `Pck`
- `necessity(Service1,Service2,(Dmin,Dmax))` means that if `Service1` is scheduled on day d_1, `Service2` should be scheduled in the interval $[d_1 + \mathtt{Dmin}, d_1 + \mathtt{Dmax}]$.
- `interdiction(Service1,Service2,Ndays)` states that `Service2` cannot be scheduled for `Ndays` after `Service1`.

The ASP program for the scheduling of packets to the available days (Listing 1) follows the classical generate and test methodology. The generation part (lines 1–6) tries to assign a date `Day` to each `Packet` within the given tolerance from the ideal date.

Listing 1. *Date assignment*

```
1  0{schedule(Pat,Pck,Day):day(Day), within_tol(Pck,Day,Ideal)}1
2  :- occurrence_to_schedule(Pat,Pck),
3    ideal_date(Pat,Pck,Ideal).

5  sched_service(Pat, Service, Day) :- schedule(Pat, Packet, Day),
6    service_in_packet(Service,Packet).
7  :~ occurrence_to_schedule(Pat,Pck),  not schedule(Pat,Pck,_).

9  :- sched_service(Pat,Srv1,Day1),  sched_service(Pat,Srv2,Day2),
10     interdiction(Srv1,Srv2,Ndays),
11     Day2 >= Day1, Day2 <= Day1 + Ndays.
12 :- sched_service(Patient,Srv1,Day1),  necessity(Srv1,Srv2,_),
13     not satisfied_necessity(Patient, Srv1, Srv2).

15 satisfied_necessity(Pat, Srv1, Srv2) :-
16   sched_service(Pat,Srv1,Day1), sched_service(Pat,Srv2,Day2),
17   necessity(Srv1, Srv2, (Dmin,Dmax)),
18   Day2 >= Day1 + Dmin, Day2 <= Day1 + Dmax.
19 satisfied_necessity(Pat,Srv1,Srv2) :-
20   sched_service(Pat, Srv1, Day1),
21   necessity(Srv1, Srv2, (Dmin,Dmax)), Day1 + Dmax > horizon.

23 :- sched_service(Pat,Srv1,Day1),  sched_service(Pat,Srv2,Day2),
24   necessity(Srv1,Srv2,(Dmin,_)), Day1<Day2, Day2<=Day1+Dmin.
```

As a packet could be not scheduled at all, the number of packets scheduled within the horizon will be maximized by the weak constraint in line 7. The IC in line 9 deals with interdiction constraints: `Srv1` and `Srv2` are two services for the same patient, and the first interdicts the second for `Ndays`. The IC in line 12 ensures that each necessity constraint is satisfied. Predicate `satisfied_necessity` declares that the necessity must be either satisfied within the horizon or assumed to be satisfied beyond it, while the previous condition $\tau(s_j) \notin [\tau(s_i), \tau(s_i) + \delta_{min})$ is dealt with by the IC in line 23.

5.2 Daily agendas

In the ASP formalization in Section 5.1, services are assigned a date, but daily agendas are not handled, that is, neither a starting time is given nor services are assigned to specific operators at the various care units. The program for the daily agendas, reported in Listing 2, uses the following input predicates, in addition to those in Section 5.1:

- `srvType`(`s`, `cu`, `dur`): the list of services, together with the care unit `cu` that can provide it and the duration `dur`;
- `shift`(`D`, `CU`, `Op`, `St`, `Dur`): for each operator `Op`, the start time `St` and the duration `Dur` of the shift are provided in each day `D`, together with the care unit `CU` the operator works in.

Listing 2. *Daily agendas*

```
1  1{start_time(Pat,S,Day,Start) : time(Start)}1 :-
2     sched_service(Pat,Day,S).
3  1{provides(Op,Pat,S,Day) : shift(Day,CU,Op,_,_),srvType(S,CU,_)}1 :-
4     sched_service(Pat,S,Day).

6  precedes(P1,S1,P2,S2,Day) :- srvType(S1,_,D1),
7     Start1 + D1 <= Start2,
8     start_time(P1,S1,Day,Start1),
9     start_time(P2,S2,Day,Start2).

11 :- not precedes(P,S1,P,S2,Day), not precedes(P,S2,P,S1,Day),
12    sched_service(P, S1, Day), sched_service(P, S2, Day),
13    S1 != S2.
14 :- not precedes(P1,S1,P2,S2,Day),
15    not precedes(P2,S2,P1,S1,Day),
16    provides(CU,Op,P1,S1,Day), provides(CU,Op,P2,S2,Day),
17    sched_service(P1,S1,Day), sched_service(P2,S2,Day), P1!=P2.

19 :- sched_service(P,S,Day), srvType(S,CU,Dur),
20    provides(Op,P,S,Day), start_time(P,S,Day,Start),
21    Start + Dur > StartShift + DurShift,
22    shift(Day,CU,Op,StartShift,DurShift).
23 :- sched_service(P,S,Day), provides(Op,P,S,Day),
24    Start < StartShift, start_time(P,S,Day,Start),
25    shift(Day,CU,Op,StartShift,DurShift).
```

In the generate part of the daily agenda program (Listing 2), each scheduled service is assigned a start time (line 2) and an operator of the care unit that provides the required service type (line 3), leveraging on the daily assignment defined by predicate `sched_service` (line 5 of Listing 1).

In order to avoid overlapping between services of the same patient (constraint *iii* of the daily agenda problem) or delivered by the same operator (see *ii*), we define predicate `precedes` (line 7), stating that service `S1` of patient `P1` precedes `S2` of `P2` if they are scheduled on the same day and `S1` terminates before or at the same time as `S2` starts. Now if two services `S1` and `S2` are for the same patient or are provided by the same care unit operator `Op` in the same `Day`, one of the two services must precede the other (ICs in lines 11 and 15). Finally, ICs in lines 19 and 23 state that each service should be scheduled within the working shift of the operator who delivers it (constraint *i* of the daily agenda).

5.3 LBB decomposition

The ASP formalization of Sections 5.1 and 5.2 correctly solves the NCD Agenda problem; on the other hand, solving such a difficult problem in a monolithic approach does not scale well with the size of the instance. To speed up the solving process while retaining the completeness of the search, we apply LBBD Hooker and Ottosson (2003).

The NCD Agenda can be cast as in equation (1) in which the MP (based on the ASP program in Section 5.1) maximizes the number of scheduled packets while assigning a day to each scheduled packet, while a series of SPs (based on the program in Section 5.2), one for each day, assign a time and an operator to each service. Decomposing the program this way provides a strong improvement, since the SPs are independent problems, one per day, and they could even be solved in parallel (although in our experiments we do not exploit such parallelism in order to have a fair comparison with the monolithic approach). On the other hand, some SPs could be infeasible, since the MP does not contain all constraints of the NCD Agenda.

The MP is an optimization problem, while each SP is a satisfiability problem. In case all SPs are satisfiable, the optimal solution of the MP is also the optimum of the whole NCD Agenda problem. Otherwise, if one of the SPs is infeasible, a no-good is returned to the MP conveying the information that the particular set of health services the MP has assigned to that day cannot be feasibly served. Then, the MP is solved again, with the additional no-good, which avoids looping. Convergence occurs when each SP admits a feasible solution. Such solution is provably optimal.

In particular, the unfeasible SP returns to the MP the set of packets that could not be scheduled on that day, as a set of facts of the form

```
unfeas_subproblem(patient,packet,day,gid)
```

together with a fact `nogood_id(gid)`, where `gid` is a unique identifier for the group of packets. A new version of the MP is then generated, appending to the previous code the new facts `unfeas_subproblem` and `nogood_id`, and together with the following IC, that avoids generating schedules for the same day including all the packets in the no-good

$$:\!-\mathtt{schedule}(\mathtt{Pat},\mathtt{Pck},\mathtt{Day}) : \mathtt{unfeas\_subproblem}(\mathtt{Pat},\mathtt{Pck},\mathtt{Day},\mathtt{Gid}); \mathtt{nogood\_id}(\mathtt{Gid}). \tag{5}$$

Example 1

Consider the example in Figure 1, already introduced in Section 4. The MP may schedule both patients on day 1, as depicted. In such a case, the SP for day 1 will detect infeasibility and return a no-good to the MP, stating that both packets cannot fit together on that day:

$$\begin{array}{l}\mathtt{unfeas\_subproblem}(\mathtt{p1},\mathtt{pck1},\mathtt{day1},\mathtt{gid1}).\\ \mathtt{unfeas\_subproblem}(\mathtt{p2},\mathtt{pck2},\mathtt{day1},\mathtt{gid1}).\quad \mathtt{nogood\_id}(\mathtt{gid1}).\end{array} \tag{6}$$

The MP receives the no-good and, in the following iterations, it will contain the IC in equation (5), which is grounded into

$$:\!-\mathtt{schedule}(\mathtt{p1},\mathtt{pck1},\mathtt{day1}),\mathtt{schedule}(\mathtt{p2},\mathtt{pck2},\mathtt{day1}).$$

so that, in the following iterations, at most one of the two packets can be assigned to day 1. Since a limited number of options are present, after a certain number of iterations yielding similar no-goods, the MP will schedule p_1 on day 3 and p_2 on any other day. The process then stops and returns a feasible solution.

In LBBD, in order to speedup convergence, it is worth strengthening the MP by adding a relaxed version of some of the SPs constraints. In our case, in the MP we avoid any

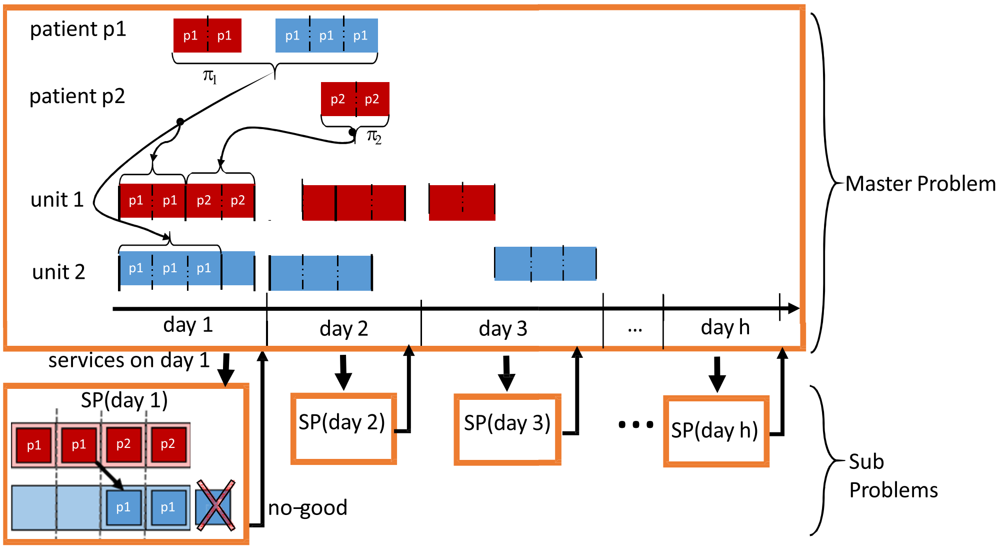

Fig. 1. Interaction scheme in LBBD.

reference to the timing within the day, but we state that, for each day and each care unit, the total duration of the services assigned to that care unit cannot exceed the sum of the shift duration of all the operators of that unit:

$$
\begin{aligned}
&\texttt{:-day(D), total\_time(D, CU, TotTime),}\\
&\texttt{\#sum\{Dur, P, S : sched\_service(P, S, D), srvType(S, CU, Dur)\}} > \texttt{TotTime.}
\end{aligned}
\tag{7}
$$

```
total_time(D, CU, Time):- srvType(_,CU,_), day(D),
    #sum{Dur,D,CU:shift(D,CU,Op, _, Dur)} = Time.
```

Note that predicate `total_time` is grounded into a set of facts by the grounder `gringo`.

The MP solution may violate the daily agenda constraints: as shown in Example 1, the MP may schedule both packets on day 1 or on day 2. However, with the availabilities depicted in Fig. 1, IC (7) forbids the MP to schedule both packets on day 3, as the red care unit provides only two time units.

5.4 *Multi-shot solving*

The interaction scheme in Section 5.3 can be implemented by a script that iteratively invokes the ASP solver on the MP, on the SPs and adds to the MP code the no-goods generated by the SPs. However, in this way, the MP should be solved from scratch at every iteration, losing information about the clauses learnt in the previous iteration.

Recent versions of Clingo Gebser *et al.* (2019) allow the customization of ASP solving processes that deal with continuously changing logic programs, called *multi-shot solving* (MS). The solving process can be controlled through commands written in Python. It is possible to integrate non-ground input rules into subprograms having a name and a list of parameters, and that are introduced by the `#program` directive. A dedicated subprogram `base` gathers all the rules that are not included in a subprogram Kaminski *et al.* (2017). By default, Clingo grounds and solves just the `base` program, but we can add control in

Python using a main routine taking as argument a control object representing the state of Clingo.

In order to exploit MS in the devised LBBD, the following subprogram is added to the ASP formulation of the MP:

```
#program nogood(pat,pck,day,gid).
unfeas_subproblem(pat,pck,day,gid). nogood_id(gid).
:- schedule(Pat,Pck,D) : unfeas_subproblem(Pat,Pck,D,Gid);
    D=day, Gid=gid.
```

The parameters `pat`, `pck`, and `day` correspond to the arguments of the `schedule` predicate, and `gid` identifies a group of schedules that produced an inconsistent SP, as in equation (5).

The `nogood` program can be grounded incrementally from a Python script, passing a list L of ground terms; each term contains the four parameters of `nogood`. In the LBBD, the parameters will be the schedules that made unfeasible one of the SPs, provided as a set of ground facts `unfeas_subproblem`.

Algorithm 2 gives the pseudocode that controls the solving process, solving in sequence the MP (*"base"*), then each SPs with an external call to Clingo for each day. Afterward, from failing SPs the group of packets scheduled by the MP in the specific day is added as a no-good. This information is added to the MP grounding the `nogood` program, and the process loops until no new constraints are added, that is, all SPs are satisfiable.

Algorithm 2 LBBD with multi-shot solving

```
function LBBD($prg$)
prg.ground([("$base$", [])])                          ▷ ground the base program (MP)
do
    $AS^{MP}$ = prg.solve()                            ▷ solve the ground program(s)
    if MP has no solution then return Unsatisfiable
    NGs = $\emptyset$
    for $day$ in $horizon$ do
        $AS^{SP_{day}}$=SOLVE_SP($day, AS^{MP}$)          ▷ solve a subproblem for each day
        if $SP_{day}$ has no solution then
            NGs = NGs ∪ COMPUTE_NOGOODS($day, AS^{MP}$)
    end for
    prg.ground(["$nogood$", $NGs$])       ▷ ground nogood program with parameters
while $NGs \neq \emptyset$
return $AS^{MP} \cup \bigcup_{day} AS^{SP_{day}}$
```

6 Experimental results

We implemented an instance generator based on well-assessed and publicly available medical guidelines for the most common NCDs[1]. The generator allocates resources on a

[1] https://salute.regione.emilia-romagna.it/cure-primarie/diabete/gestione-integrata-del-diabete-mellito-di-tipo-2-2017/, https://www.regione.toscana.it/documents/10180/23793180/ALL+A+23-2019+PDTA-Diabete.pdf/f1e8ea87-145f-08c4-6c3d-16b69f5f43c2?t=1578658143393/

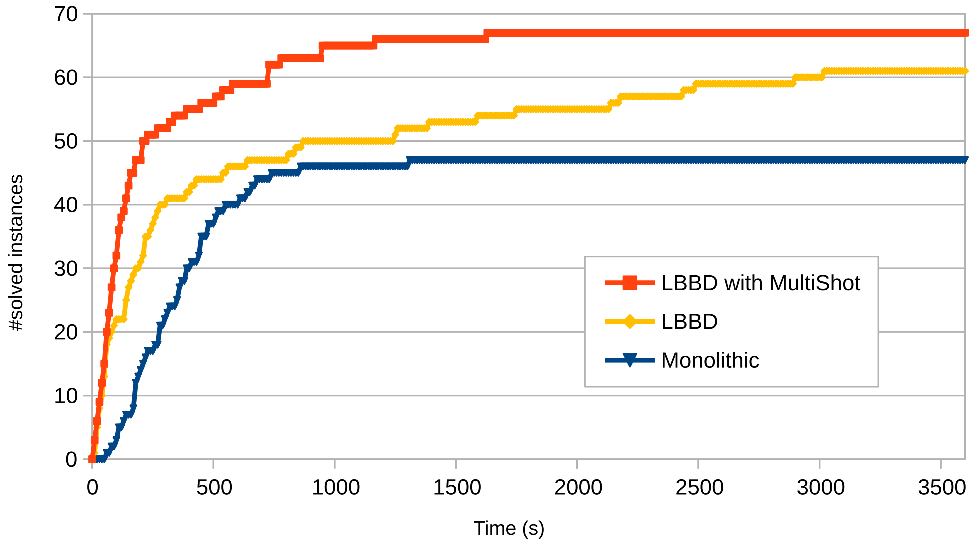

Fig. 2. Number of solved instances vs running time.

weekly basis and replicates the allocation in each week of the given time horizon, then it generates CPs. Specifically, we have: (i) five CUs, each of them with a daily capacity (expressed in number of slots) drawn with uniform probability in the range [24, 60]; (ii) a number of operators drawn with uniform probability in [1,4] on each day of the week for each CU; (iii) services with a duration (expressed in number of slots) in [6,15] and associated with a CU; (iv) packets made of four services at most; and (v) a given number of patients with a number of CPs in [1,4]. The probability of assigning a number of CPs to a given patient is inversely proportional to the number of CPs itself.

For each number of patients in $\{10, 20, 40\}$ and length of the planning horizon in $\{30,60\}$ days, 20 instances are generated, summing up to 120 instances. Experiments were run with Clingo 5.6.2 with a time limit of 1 h on a Ubuntu 22.04.1 LTS OS, Intel(R) Xeon(R) CPU E5-2430 v2 @ 2.50GHz machine with a 32GB System Memory.

Figure 2 shows on the y-axis the number of instances solved by each of the three approaches within a given computation time reported on the x-axis. The decomposition approach lets one solve to optimality between 30% and 42% more instances.

To have a finer detail on how the running time varies with the size of the instance, we plot in Figure 3 the runtime versus the number of services to be scheduled, comparing the monolithic approach and the LBBD method equipped with MS. Solid lines represent total time, while dotted lines show the time required by grounding. Instances running into out of memory were counted as running for 3600s. For the LBBD-MS we also show the time for the overall algorithm (orange series) as well as the time required by the MP (green series). The time required by the SP can be easily evaluated by difference between the orange and the green series.

We can observe the following facts: (i) as expected, the total time grows as the number of services increases regardless of the method used; (ii) for the monolithic approach, the running time is almost entirely spent in the grounding phase; (iii) for the LBBD with MS the grounding time spent in the MP is stable across instances and negligible; the grounding time of this approach is almost all due to the SPs.

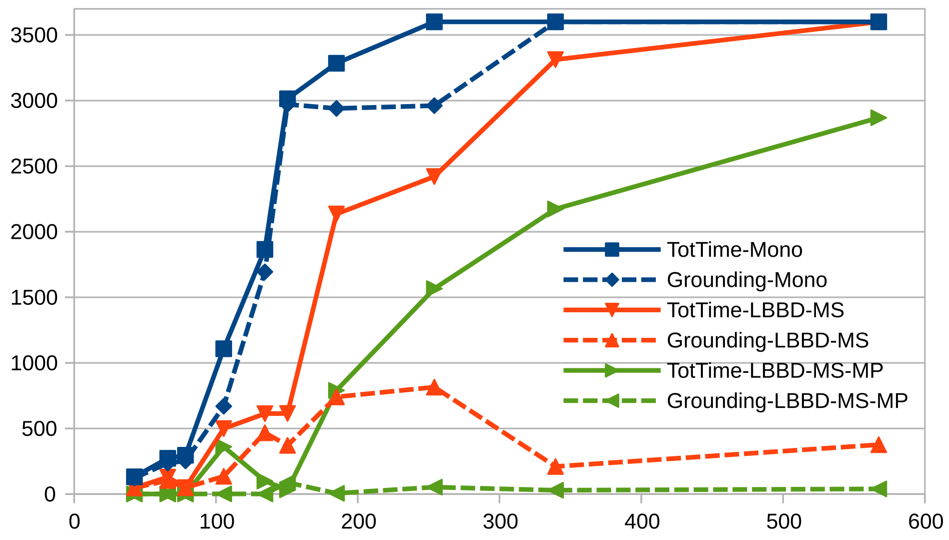

Fig. 3. Run time vs number of services – all instances.

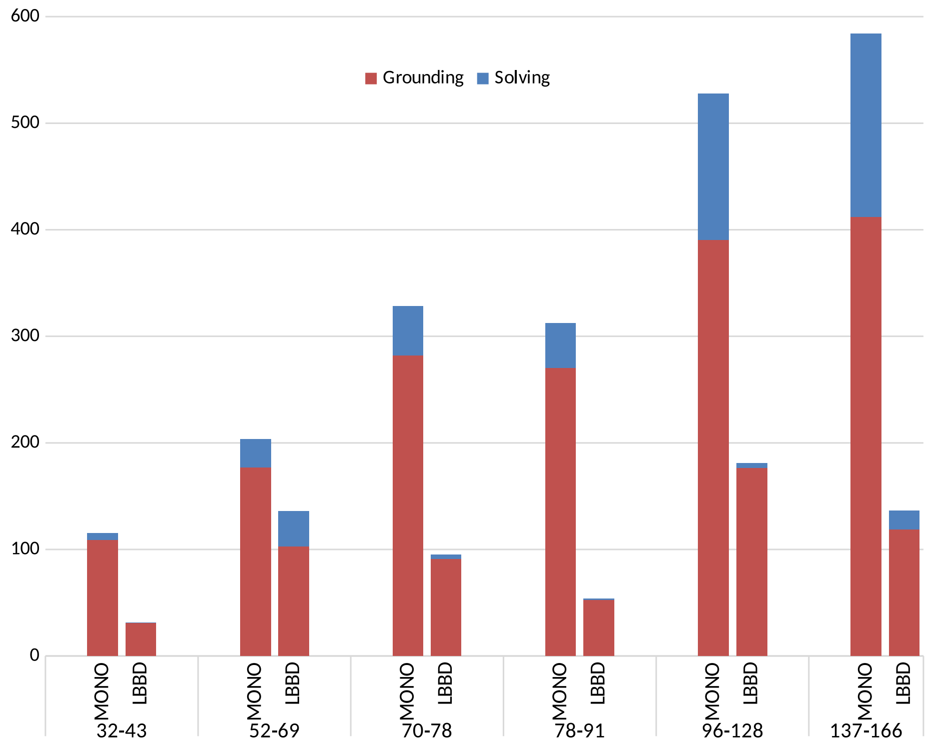

Fig. 4. Run time vs number of services – instances without timeout.

To show the exact speedup in the two phases, Figure 4 considers only those instances for which the monolithic approach was able to terminate within the timeout; clearly, these are the most favorable for the monolithic approach. In these instances, the average grounding time for LBBD was 39.2% of that of the monolithic, while the average solving time was

12.05% of that of the monolithic, with almost an order of magnitude of improvement in the solving time.

7 Conclusions

In this work, we adopted LBBD in a solving process based on ASP; to the best of our knowledge, this is the first work adopting this methodology with ASP. LBBD is a widely used solving technique in Operations Research and in Constraint Programming, and it constitutes one of the most effective technologies for hybridization of Integer Linear Programming and Constraint Programming technologies. With LBBD, the problem can be decomposed without losing completeness, that is, maintaining the possibility to find the optimal solution and prove its optimality. Even more interestingly, this opens a new avenue of integration of ASP with other paradigms for solving constrained optimization problems, for example, new hybrid algorithms involving ASP and Constraint Programming or Integer Linear Programming.

The considered application is a scheduling problem for chronic outpatients with NCDs needing recurrent services at the hospital. The experimental results show that LBBD enlarges the applicability of ASP to larger instances without sacrificing optimality. Future work includes strengthening the efficiency of the LBBD scheme by providing stronger no-goods from the subproblems to the MP and to apply LBBD to other problems.

We believe that the LBBD approach could be applied to a number of applications already available for ASP (e.g., the applications presented by Guido *et al.* (2020), Cardellini *et al.* (2021), and Caruso *et al.* (2023), just to name a few in the healthcare domain) in which the global problem was greedily split into subparts; we hope that this work could be of inspiration for the many ASP applications in which the authors forewent obtaining optimality and widen even further the ASP applications in the real world.

Competing interests

The authors declare none.

Acknowledgments

This work was partially supported by GNCS-INdAM.

References

Alviano, M., Bertolucci, R., Cardellini, M., Dodaro, C., Galatà, G., Khan, M. K., Maratea, M., Mochi, M., Morozan, V., Porro, I. and Schouten, M. 2020. Answer set programming in healthcare: Extended overview. *Italian workshop on Planning and Scheduling 2021 – International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion 2021 @ AI*IA 2745.*

Alviano, M., Dodaro, C. and Maratea, M. 2018. Nurse (re) scheduling via Answer Set Programming. *Intelligenza Artificiale 12,* 2, 109–124.

Benini, L., Lombardi, M., Milano, M. and Ruggiero, M. 2011. Optimal resource allocation and scheduling for the CELL BE platform. *Annals of Operations Research 184,* 1, 51–77.

Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Maratea, M., Ricca, F. and Schaub, T. 2020. ASP-Core-2 input language format. *Theory and Practice of Logic Programming 20,* 2, 294–309.

Cappanera, P., Gavanelli, M., Nonato, M. and Roma, M. 2022. A decomposition approach to the clinical pathway deployment for chronic outpatients with comorbidities. In *Optimization in Artificial Intelligence and Data Sciences: ODS, First Hybrid Conference, Rome, Italy, September 14-17, 2021*, L. Amorosi, P. Dell'Olmo, and I. Lari, Eds. AIRO Springer Series. Springer International Publishing, Cham, 213–226.

Cappanera, P., Gavanelli, M., Nonato, M. and Roma, M. 2023. Decomposition approaches for scheduling chronic outpatients' clinical pathways in answer set programming. *Journal of Logic and Computation.* https://doi.org/10.1093/logcom/exad038.

Cardellini, M., De Nardi, P., Dodaro, C., Galatà, G., Giardini, A., Maratea, M. and Porro, I. 2021. A two-phase ASP encoding for solving rehabilitation scheduling. In *Rules and Reasoning: 5th International Joint Conference, RuleML+ RR 2021, Leuven, Belgium, September 13–15, 2021, Proceedings*, S. Moschoyiannis, R. Peñaloza, J. Vanthienen, A. Soylu, and D. Roman, Eds. Springer, Cham, 111–125.

Caruso, S., Galatà, G., Maratea, M., Mochi, M. and Porro, I. 2023. Scheduling pre-operative assessment clinic with answer set programming. *Journal of Logic and Computation.* https://doi.org/10.1093/logcom/exad017.

Dal Palù, A., Dovier, A., Formisano, A. and Pontelli, E. 2018. ASP applications in bio-informatics: A short tour. *Künstliche Intelligenz 32,* 2–3, 157–164.

Dodaro, C., Galatà, G., Grioni, A., Maratea, M., Mochi, M. and Porro, I. 2021. An ASP-based solution to the chemotherapy treatment scheduling problem. *Theory and Practice of Logic Programming 21,* 6, 835–851.

Dodaro, C., Galatà, G., Khan, M. K., Maratea, M. and Porro, I. 2022. Operating room (re)scheduling with bed management via ASP. *Theory and Practice of Logic Programming 22,* 2, 229–253.

Dodaro, C. and Maratea, M. 2017. Nurse scheduling via answer set programming. In *International Conference on Logic Programming and Nonmonotonic Reasoning.* Springer, Springer International Publishing, Cham, 301–307.

El-Kholany, M. M. S., Gebser, M. and Schekotihin, K. 2022. Problem decomposition and multi-shot ASP solving for job-shop scheduling. *Theory and Practice of Logic Programming 22,* 4, 623–639.

El-Kholany, M. M. S., Schekotihin, K. and Gebser, M. 2022. Decomposition-based job-shop scheduling with constrained clustering. In *Practical Aspects of Declarative Languages*, J. Cheney and S. Perri, Eds. LNCS, vol. 13165. Springer International Publishing, Cham, 165–180.

Erdem, E., Gelfond, M. and Leone, N. 2016. Applications of answer set programming. *AI Magazine 37,* 3, 53–68.

Falkner, A. A., Friedrich, G., Schekotihin, K., Taupe, R. and Teppan, E. C. 2018. Industrial applications of answer set programming. *Künstliche Intelligenz 32,* 2–3, 165–176.

Fazel-Zarandi, M. M. and Beck, J. C. 2009. Solving a location-allocation problem with logic-based Benders' decomposition. In *Principles and Practice of Constraint Programming 2009*, I. P. Gent, Ed. LNCS, vol. 5732. Springer, Berlin, Heidelberg, 344–351.

Galatà, G., Maratea, M., Mochi, M., Morozan, V. and Porro, I. 2021. An ASP-based solution to the operating room scheduling with care units. In *Proc. Italian workshop on Planning and Scheduling 2021 - International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion 2021 @ AI*IA*, R. D. Benedictis, M. Maratea, A. Micheli, E. Scala, I. Serina, M. Vallati, and A. Umbrico, Eds. CEUR Workshop Proceedings, vol. 3065. CEUR-WS.org.

Gebser, M., Kaminski, R., Kaufmann, B. and Schaub, T. 2019. Multi-shot ASP solving with clingo. *Theory and Practice of Logic Programming 19,* 1, 27–82.

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In *International Conference on Logic Programming*, R. A. Kowalski and K. A. Bowen, Eds. MIT Press, 1070–1080.

Guido, R., Ielpa, G. and Conforti, D. 2020. Scheduling outpatient day service operations for rheumatology diseases. *Flexible Services and Manufacturing Journal 32,* 1, 102–128.

Hooker, J. and Ottosson, G. 2003. Logic-based Benders decomposition. *Mathematical Programming, Ser. A 96*, 33–60.

Hooker, J. N. 2019. *Logic-Based Benders Decomposition for Large-Scale Optimization.* Springer International Publishing, Cham, 1–26.

Kaminski, R., Schaub, T. and Wanko, P. 2017. *A Tutorial on Hybrid Answer Set Solving with clingo.* Springer International Publishing, Cham, 167–203.

Kifer, M. and Liu, Y. A. 2018. *Declarative logic programming: theory, systems, and applications.* Vol. 20. Association for Computing Machinery and Morgan & Claypool.

Leutwiler, F. and Corman, F. 2022. A logic-based Benders decomposition for microscopic railway timetable planning. *European Journal of Operational Research 303,* 2, 525–540.

Lifschitz, V. and Turner, H. 1994. Splitting a logic program. In *International Conference on Logic Programming*, P. Van Hentenryck, Ed. MIT Press, 23–37.

Riise, A., Mannino, C. and Lamorgese, L. 2016. Recursive logic-based Benders' decomposition for multi-mode outpatient scheduling. *European Journal of Operational Research 255*, 719–728.

Schüller, P. 2018. Answer set programming in linguistics. *Künstliche Intelligenz 32,* 2-3, 151–155.

Zhu, X., Son, J., Zhang, X. and Wu, J. 2023. Constraint programming and logic-based Benders decomposition for the integrated process planning and scheduling problem. *Omega 117*, 102823.