Ann Oper Res (2009) 171: 27–43
DOI 10.1007/s10479-008-0383-5

On the separability of subproblems in Benders decompositions

Marco Cadoli · Fabio Patrizi

Published online: 21 June 2008
© Springer Science+Business Media, LLC 2008

Abstract Benders decomposition is a well-known procedure for solving a combinatorial optimization problem by defining it in terms of a *master problem* and a *slave problem*. Its effectiveness relies, among other factors, on the possibility of synthesizing *Benders cuts* that rule out not only one, but a large class of trial values for the master problem. In turn, for the class of problems we consider (i.e., optimization plus constraint satisfaction) the possibility of *separating* the slave problem into several subproblems—i.e., problems exhibiting strong intra-relationships and weak inter-relationships—can be exploited for improving searching procedures efficiency. The notion of *separation* is typically given informally, or relying on syntactical aspects. This paper formally addresses the notion of slave problem separability by giving a semantic definition and exploring it from the computational point of view. Several examples of separable problems are provided, including some proving that a semantic notion of separability is much more helpful than a syntactic one. We show that separability can be formally characterized as equivalence of logical formulae, and prove the undecidability of the separability check problem. Finally, we show how there are cases where automated tools can still be used for checking subproblem separability.

Keywords Benders decomposition · Constraint programming · Problem separation

1 Introduction and motivations

Benders decomposition (Benders 1962) is a well-known procedure for solving combinatorial optimization problems, which relies on the idea of distinguishing *primary* from *secondary* variables, defining a *master problem* over primary variables and a *slave problem* over

An earlier version of this paper appeared in *Proc. of 3rd International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* (CP-AI-OR'06). Cork, Ireland. May, 2006.

M. Cadoli · F. Patrizi (✉)
SAPIENZA—University of Rome, Via Ariosto 25, 00185 Rome, Italy
e-mail: patrizi@dis.uniroma1.it

M. Cadoli
e-mail: cadoli@dis.uniroma1.it

secondary variables, given a trial value for primary variables. Every unsuccessful attempt to solve the slave problem is recorded as a *Benders cut* and added to the master problem, until an optimal solution is found, or the problem is proven to be infeasible.

Here, we deal with the class of "optimization plus constraint satisfaction" (OPT + CS) problems (Hooker 2000). In this context, two important factors which contribute to make the above procedure effective are: (1) the possibility of using different technologies for solving the master and the slave problem, e.g., ILP and CP, respectively (Hooker and Ottosson 2003; Jain and Grossmann 2001), and (2) the possibility of synthesizing Benders cuts that rule out not only one, but a large class of trial values for the master problem. In this paper, we focus on the second factor, and specifically on the notion of *separability* of the slave problem, which intuitively means that it can be formulated using several subproblems exhibiting strong intra-relationships and weak inter-relationships. As a matter of fact, it has been noted in Hooker and Ottosson (2003), Hooker (2000), Cambazard and Jussien (2005) that if the slave problem is separable then it is possible to design a Benders cut (or *nogood*) that excludes several instantiations of the primary variables or, in other words, a nogood which is a partial, and not a total, assignment to primary variables. Therefore, the ability to recognize slave problem separability is a very important factor for the efficiency of a Benders decomposition. In fact, previous work (Hooker and Ottosson 2003; Hooker 2000; Cambazard and Jussien 2005) assumes the slave problem is modeled by a set of easier (sub)problems.

Let us introduce our running example, taken from Hooker and Ottosson (2003), Hooker (2000), which refers to a *machine scheduling* problem.

Example 1.1 (Machine scheduling problem (Hooker and Ottosson 2003; Hooker 2000)) Machine Scheduling is the problem of finding an assignment of a set of jobs to a set of machines in such a way that (1) constraints on release and (2) due date are satisfied, (3) machines are single-task, and a cost function is minimized. Using the modelling language of the OPL system (Van Hentenryck 1999), one possible model is as follows:

```
// INPUT DESCRIPTION
{int+} Jobs=...;                  // The set of jobs to be scheduled
int+ horizon=...;                 // Max start time for jobs
int+ n_machines=...;              // Number of machines
range Time[1..horizon];           // Range "Time" definition
range Machines[1..n_machines];    // Range "Machines" definition
int+ ReleaseDate[Jobs]=...;       // Each job has a release date
int+ DueDate[Jobs]=...;           // Each job has a due date
int+ Cost[Jobs,Machines]=...;     // Machines incur different costs per job
int+ Duration[Jobs,Machines]=...;// Machines run at different speeds per job
// SEARCH SPACE
var Machines Assignment[Jobs];
var Time StartTime[Jobs];
// OBJECTIVE FUNCTION
minimize
   sum (j in Jobs) Cost[j,Assignment[j]]
// CONSTRAINTS
subject to {
   forall (j in Jobs) // 1. RESPECT RELEASE DATE
      StartTime[j] >= ReleaseDate[j];
   forall (j in Jobs) // 2. RESPECT DUE DATE
      StartTime[j] + Duration[j,Assignment[j]] <= DueDate[j];
   forall (t in Time) // 3. MAX ONE JOB PER MACHINE AT EACH TIME POINT
     forall (m in Machines)
       sum (j in Jobs)
         (Assignment[j] = m &
```

```
          StartTime[j] <= t < (StartTime[j] + Duration[j,Assignment[j]])
        ) <= 1;
};
```

The Benders decomposition suggested in Hooker and Ottosson (2003) selects `Assignment` as primary and `StartTime` as secondary variables. Moreover, it defines the master problem as an unconstrained minimization problem, and the slave problem[1] as a decision problem on constraints 1, 2, and 3 (for a given optimal instantiation of `Assignment` and ignoring the objective function). A given optimal instantiation $\overline{\texttt{Assignment}}$ which is infeasible for the slave problem is called a *Benders cut* (or *nogood*), and the next iteration of the master problem includes the constraint $\texttt{Assignment} \neq \overline{\texttt{Assignment}}$, until a feasible instantiation is found, or the problem is proven to be infeasible.

This is a "raw" version of the decomposition, which ignores the fact that the slave problem is *separable* with respect to the machines. As an example, if we have six jobs and three machines, we can consider a separate (slave) subproblem for each machine. If $\overline{\texttt{Assignment}} = \texttt{[1,1,2,1,3,2]}$ is optimal for the master problem, and no serial schedule of jobs `3,6` on the second machine exists, we can safely add the constraint

```
Assignment[3] <> 2 \/ Assignment[6] <> 2.
```

Constraints of this kind rule out a whole set of assignments (not just one; in this case $3^{(6-2)} = 81$) to the primary variables, and can be added for each infeasible subproblem. This ultimately results in a more efficient decomposition.

Usually (cf., e.g., Cambazard and Jussien 2005; Hooker 2000), when dealing with OPT + CS problems, slave problems are assumed separable and their subproblems are defined manually. The issue of stating *when* problems actually separate has not, to the best of our knowledge, been addressed, even though it might be helpful, as we show throughout the paper, for automatically detecting or, possibly, synthesizing effective Benders decompositions. An informal notion of separability is typically used in the literature, but we claim that the importance of this concept calls for precise definitions and careful analysis. In what follows we elaborate the running example, to substantiate our claim.

Example 1.2 (Example 1.1, continued) Consider constraint 3 of Example 1.1. Since (i) it is universally quantified with respect to machines, (ii) constraint on machine `m` involves no other machines and (iii) it is the only constraint involving machines—in other words, the syntactic form of the problem leads to conclude that the problem can be rewritten as a conjunction of constraints, pairwise referring to different machines—then one can claim problem separability with respect to machines. The methodological problem with this syntax-based, sufficient criterion is that it depends heavily on the specific way the problem is formulated (cf. also forthcoming Example 3.3). To see this point, consider the following statement, equivalent to constraint 3.

```
// 3'. NO TWO JOBS RUNNING ON THE SAME MACHINE AT EACH TIME POINT
forall(t in Time)
  sum (i,j in Jobs: i <> j)
      ( Assignment[i] = Assignment[j] &
        StartTime[i] <= t < (StartTime[i] + Duration[i,Assignment[i]]) &
        StartTime[j] <= t < (StartTime[j] + Duration[j,Assignment[j]])
      ) = 0;
```

[1] In Sect. 3, such problem will be shown to belong, in general, to the complexity class NP (Fagin 1974; Papadimitriou 1994).

Now, previous criterion no longer applies (though the problem still separates) and a more general, possibly semantic, one is needed in order to check for separability.

It is everyday experience that different formulations, all of them being intuitive, can be done for a problem, as Example 1.3 shows, sometimes in the hope of obtaining models with better performance. Thus, independence of separability checking criteria from specific problem formulation is a very desirable property.

Example 1.3 (Example 1.1, continued) We can define a dependent array `RunsOn` storing for each time point and each job the machine that runs the job (or a negative number if the job is not running). In fact, in this way we can define the "single-task machines" constraint (3 or 3′) by means of a global `alldifferent` constraint, just stating that running machines are all different at each time point. The `alldifferent` constraint often performs very well (Puget 1998), especially in connection with "channelling constraints" (Walsh 2001).

```
range MachinesPlus[-card(Jobs)..n_machines]; // negative numbers: job not running
var MachinesPlus RunsOn[Time,Jobs];
// DEFINITION OF RunsOn:
forall (t in Time)
  forall (j in Jobs){
    (StartTime[j] <= t < (StartTime[j] + Duration[j,Assignment[j]])
     => RunsOn[t,j] = Assignment[j])
    &
    (StartTime[j] > t \/ t >= (StartTime[j] + Duration[j,Assignment[j]])
     =>  RunsOn[t,j] = -j); // negative numbers are all different
  };
// 3''. AT EACH TIME POINT RUNNING MACHINES ARE ALL DIFFERENT
forall (t in Time)
  alldifferent (all (j in Jobs) RunsOn[t,j]);
```

Again, constraint 3″ is not universally quantified with respect to the machines, but it nevertheless separates.

In this paper, we investigate the possibility of automating the process of checking slave problem separability in the context of Benders decompositions of OPT + CS problems. In particular, given a problem *PB* and applying a given Benders decomposition schema which leads to a constraint satisfaction slave problem (Hooker 2000), our goal is to state the conditions, if any, that make the slave problem separable. To this end, we first address the notion of slave problem separability by giving a semantic definition, and then we explore it from the computational point of view, providing two theorems which show that (i) separability can be formally characterized as equivalence of logical formulae and (ii) the problem of checking separability is not decidable. Nonetheless, in an example we show how the formal characterization we provide is useful for exploiting automated tools in checking separability, even though their application gives no guarantee of completeness.

The exposition is structured as follows. In Sect. 2 we recall the definition of Benders decomposition, in Sect. 3 a formal definition of separation is given, in Sect. 4 we show semantic and computational characterizations and, finally, Sect. 5 draws some conclusions.

2 Preliminaries

Given two arrays of variables $\boldsymbol{p} = (p_1, \ldots, p_n)$ (primary) and $\boldsymbol{s} = (s_1, \ldots, s_m)$ (secondary) which may take values, respectively, from sets $P = C_1^p \times \cdots \times C_n^p$ and $S = C_1^s \times \cdots \times C_m^s$,

in this paper we consider problems of the form:

$$PB: \begin{cases} \min\{f(\boldsymbol{p})\} & \text{objective function (o.f.)} \\ \text{s.t.} & \\ \quad \alpha(\boldsymbol{s}) & \text{constraint 1 (c1)}, \\ \quad \gamma(\boldsymbol{p}) & \text{constraint 2 (c2)}, \\ \quad \beta(\boldsymbol{p},\boldsymbol{s}) & \text{constraint 3 (c3)}, \\ \quad \boldsymbol{p} \in P & \text{primary variables domain (p.v.d.)}, \\ \quad \boldsymbol{s} \in S & \text{secondary variables domain (s.v.d.)}, \end{cases} \tag{1}$$

where α, γ and β are suitable representations of constraints in which, respectively, only $\boldsymbol{s}$ variables, only $\boldsymbol{p}$ variables, or both occur. In Hooker and Ottosson (2003), Cambazard and Jussien (2005) generalizations of the above problem in which, e.g., variables from $\boldsymbol{s}$ may occur in the objective function, are studied. According to Hooker and Ottosson (2003), problems of the form (1) can be solved by applying a "logic-based" Benders decomposition schema that gives raise to the following problems:

$$MP^k: \begin{cases} \min\{f(\boldsymbol{p})\} & \text{o.f.} \\ \text{s.t.} & \\ \quad \gamma(\boldsymbol{p}) & \text{(c2)}, \\ \quad CUT_{\boldsymbol{p}^i}(\boldsymbol{p}) & \\ \quad (i = 1, \ldots, k-1) & \text{Benders cuts}, \\ \quad \boldsymbol{p} \in P & \text{p.v.d.}, \end{cases} \tag{2}$$

$$SP: \begin{cases} \alpha(\boldsymbol{s}) & \text{(c1)}, \\ \beta(\overline{\boldsymbol{p}},\boldsymbol{s}) & \text{(c3)}, \\ \boldsymbol{s} \in S & \text{s.v.d.} \end{cases} \tag{3}$$

Master Problem (MP^k) is the problem of finding an assignment to $\boldsymbol{p} \in P$ that minimizes the objective function $f(\boldsymbol{p})$ while satisfying (i) $\gamma(\boldsymbol{p})$, and (ii) the Benders cuts $CUT_{\boldsymbol{p}^i}(\boldsymbol{p})$ $(i = 1, \ldots, k-1)$ generated at the previous $k-1$ iterations. When $k = 1$, MP^1 contains no cut, and the decomposition is just a bipartition of the constraints of *PB* into (i) those over variables involved in the objective function, put into MP^1, and (ii) the remaining ones, belonging to *SP*.

Slave Problem (SP) is the feasibility problem of checking whether there exists an assignment $\overline{\boldsymbol{s}}$ that, along with a given assignment $\overline{\boldsymbol{p}}$ obtained as solution of MP^k, satisfies the constraints $\alpha(\boldsymbol{s})$ and $\beta(\overline{\boldsymbol{p}},\boldsymbol{s})$. If such $\overline{\boldsymbol{s}}$ exists then $(\overline{\boldsymbol{p}},\overline{\boldsymbol{s}})$ is a solution to *PB*, otherwise, problem MP^{k+1} is generated by adding to MP^k a Benders cut $CUT_{\boldsymbol{p}^k}(\boldsymbol{p})$.

Referring to Example 1.1, $\boldsymbol{p}$ is `Assignment`, $\boldsymbol{s}$ is `StartTime`, $\alpha(\boldsymbol{s})$ is constraint "`1. RESPECT RELEASE DATE`", $\beta(\boldsymbol{p},\boldsymbol{s})$ is the conjunction of constraints "`2. RESPECT DUE DATE`" and "`3. MAX ONE JOB PER MACHINE AT EACH TIME POINT`", and $\gamma(\boldsymbol{p})$ is a tautology.

One obvious desirable quality of Benders cuts is *soundness*, i.e., the guarantee that the above algorithm finds an optimal solution to *PB* for each instance. As an example, the constraint

$$CUT_{\boldsymbol{p}^k}(\boldsymbol{p}) \doteq (\boldsymbol{p} \neq \boldsymbol{p}^k), \tag{4}$$

where $\boldsymbol{p}^k$ is the solution to MP^k, is sound. The problem with (4) is that an unacceptably large number of cuts may be added to the master problem, and this may reflect in inefficiency (cf.

Example 1.1). In the next sections we look for conditions which may be helpful for having a significantly lower number of cuts.

3 Separation into subproblems

Before formalizing the notion of separability introduced in Sect. 1, we need to clarify the role played by the selection of relevant input data. Referring to Example 1.1, every choice of the machine on the solution of the master problem induces a selection of the release and due dates, costs, and durations. As an example, if $\overline{\texttt{Assignment}} = \texttt{[1,1,2,1,3,2]}$ and machine 2 is selected, then we only need the third and the sixth rows of input arrays `ReleaseDate` and `DueDate`. Analogously, we need only some entries of the `Cost` and `Duration` arrays.

In general, given a representation $\boldsymbol{R}$ of the instance, e.g., as a relational database over the schema $\mathcal{R}$, and an integer q representing the number of subproblems, we assume that there is a function $\sigma_I : \mathcal{R} \times [1, q] \to \mathcal{R}$ that *selects* the input data relevant for the i-th subproblem ($1 \le i \le q$).

Analogously, we need a way to select the variables relevant to the i-th subproblem. As an example, for the given $\overline{\texttt{Assignment}}$ and machine 2, we want to assign a `StartTime` just to jobs `3` and `6`. In general, we assume that there is a function σ_V that partitions the variables into q subsets, one for each subproblem. For the sake of simplicity, we assume that all the variables may take a value from the same set.

From now on, we represent feasibility problems like (3) with the following notation

$$\psi(\boldsymbol{R}) = \exists F : D \to C \quad \text{s.t.} \quad \phi(\boldsymbol{R}, F), \tag{5}$$

where $\boldsymbol{R}$ is a representation of the instance over the schema $\mathcal{R}$, F is the required assignment to the variables, D and C are the variables and their assignments, respectively, and $\phi(\boldsymbol{R}, F)$ is a representation of the constraints. We prefer the above notation over the notation as in (3) because it highlights the input, which is crucial for our purposes. Moreover it is worth reminding that, if C is finite and ϕ is a formula in first-order logic, then formulae of the kind (5) can represent every problem in the complexity class NP (Fagin 1974; Papadimitriou 1994). Finally, we note that there is a direct correspondence between the above notation and state-of-the-art modelling languages such as OPL. As an example, an array of variables like `StartTime` in Example 1.1 corresponds to the existentially quantified function F in (5).

Definition 3.1 (Subproblems) Given a problem ψ of the form (5), an integer $q \ge 1$ and two functions $\sigma_I : \mathcal{R} \times [1, q] \to \mathcal{R}$ and $\sigma_V : [1, q] \to 2^D$ such that $\{D_1, \ldots, D_q\}$ ($\forall i \in 1..q,\ D_i \doteq \sigma_V(i)$) is a partition of D, the following q problems are defined as the *subproblems of ψ with respect to σ_I and σ_V*:

$$\psi_i(\boldsymbol{R}) = \exists \boldsymbol{R}_i, F_i : D_i \to C \quad \text{s.t.} \quad \boldsymbol{R}_i = \sigma_I(\boldsymbol{R}, i) \wedge \phi(\boldsymbol{R}_i, F_i) \quad (i = 1, \ldots, q).$$

Definition 3.1 can be used to obtain the subproblems in a syntactical way, by means of a symbolic manipulation of the model of the problem. To see intuitively how the subproblems are obtained, we resort again to the running example.

Example 3.1 (Example 1.1, continued) Given an instance of the problem with q machines, and a value for `Assignment`, we consider the (sub)problem defined as the conjunction

of constraints 1, 2, and 3, and no objective function. As mentioned before, σ_I takes a machine `i` and the input, e.g., arrays `ReleaseDate`, `DueDate`, `Cost`, and `Duration`, and gives new arrays `ReleaseDate_i`, `DueDate_i`, `Cost_i`, and `Duration_i`. σ_I can be represented by means of simple constraints, the following being an example for `i = 1`:

```
// INPUT:
{int+} Jobs=...; int+ horizon=...; int+ n_machines=...;
range Machines [1..n_machines];
int+ ReleaseDate[Jobs]=...; int+ DueDate[Jobs]=...;
int+ Cost[Jobs,Machines]=...; int+ Duration[Jobs, Machines]=...;
// JOBS ASSIGNMENT:
Open Machines Assignment[Jobs];
// CONSTANTS DEFINITION:
int+ maxTime = max(j in Jobs)(DueDate[j]);
int+ maxCost = max(j in Jobs, m in Machines)(Cost[j,m]);
int+ maxDuration = max(j in Jobs, m in Machines)(Duration[j,m]);
// OUTPUT:
{int+} Jobs_1={j | j in Jobs: Assignment[j]=1};
var Machines n_machines_1 in 1..1; // n_machines_1 = 1
var int+ horizon_1 in horizon..horizon; // horizon_1 = horizon
var int+ ReleaseDate_1[Jobs_1] in 0..horizon;
var int+ DueDate_1[Jobs_1] in 0..maxTime;
var int+ Cost_1[Jobs_1,[1..1]] in 0..maxCost;
var int+ Duration_1[Jobs_1,[1..1]] in 0..maxDuration;
// CONSTRAINTS:
solve{
   forall(j in Jobs_1){
      ReleaseDate_1[j] = ReleaseDate[j];
      DueDate_1[j] = DueDate[j];
      Cost_1[j,1] = Cost[j,1];
      Duration_1[j,1] = Duration[j,1];
   }
};
```

Note that, coherently with Definition 3.1 where arrays $\boldsymbol{R}_i$ are existentially quantified, all items of the form `xxx_1`, e.g. `DueDate_1` and `horizon_1`, are variables that must be assigned, `Jobs_1` being a syntactical exception, due to implementation reasons, that can be yet conceptually regarded as a variable.

The other function σ_V takes a machine `i` and the secondary variables, i.e., array `StartTime`, and gives a new array of variables `StartTime_i`. The representation of σ_V is also simple, and is omitted for brevity.

Each subproblem can be simply represented by defining all constraints on the new symbols, e.g., by writing `DueDate_1` instead of `DueDate` for the first subproblem. It is worth noting that this can be done for all versions of the machine scheduling problem, i.e., for Examples 1.1, 1.2, and 1.3.

Given an instance $\boldsymbol{R}$ of a problem of the form (5), we denote as $SOL(\psi(\boldsymbol{R}))$ the set of solutions to $\psi(\boldsymbol{R})$, i.e., of the set of functions which satisfy the constraints. The following definition tells us how to integrate the solutions of the subproblems.

Definition 3.2 (Composition of solutions) Given a problem $\psi(\boldsymbol{R})$ and its q subproblems $\psi_i(\boldsymbol{R})$ as in Definition 3.1, we define the *composition* ($\bowtie$) *of the solutions* $SOL(\psi_i(\boldsymbol{R}))$ of the subproblems as follows:

$$\bowtie_{i=1}^{q} SOL(\psi_i(\boldsymbol{R})) \doteq \{F : D \to C \text{ s.t. } \forall i = 1, \ldots, q \; F|_{D_i} \in SOL(\psi_i(\boldsymbol{R}))\},$$

where $F|_{D_i}$ denotes the selection of the assignments of F to the variables in D_i.

Now we need a way to relate a problem to its subproblems, which is *semantic*, i.e., based on the respective solutions. The following definition tells us that a problem is separated by (σ_I, σ_V) if its solutions can be obtained just by composing the solutions of its subproblems.

Definition 3.3 (Separation) Given a problem of the form (5) and its q subproblems $\psi_i(\boldsymbol{R})$ as in Definition 3.1, ψ is (σ_I, σ_V)-*separated into the q problems* $\psi_1, \ldots, \psi_q$ iff

$$\forall \boldsymbol{R} \in \mathcal{R} \quad \bowtie_{i=1}^{q} SOL(\psi_i(\boldsymbol{R})) = SOL(\psi(\boldsymbol{R})).$$

Referring again to the slave problem in the three versions of Examples 1.1, 1.2 and 1.3, it is possible to see that it is (σ_I, σ_V)-separated into q subproblems according to Definition 3.3, where q is the number of machines.

Of course, problems may not separate, as shown by the next example.

Example 3.2 We add to the constraints of Example 1.1 a further constraint which avoids more than two machines running at the same time, useful, e.g., to reduce noise or energy consumption.

```
forall (t in Time) // 4. MAX TWO JOBS RUNNING AT EACH TIME POINT
    sum (j in Jobs)(
      StartTime[j] <= t < (StartTime[j] + Duration[j,Assignment[j]]
    ) <= 2;
```

The latter constraint is added to the slave problem, and the master problem is unchanged. With functions σ_I and σ_V as defined in Example 3.1, it is possible to see that the current version of the slave problem is not (σ_I, σ_V)-separated. We do that by (1) exhibiting an instance, (2) solving separately the subproblems obtained applying Definition 3.1, (3) composing their solutions according to Definition 3.2, and (4) showing that a solution which does not satisfy the original problem arises.

The instance is as follows:

```
Jobs = {1,2,3,4,5,6}; n_machines = 3; horizon = 15;
ReleaseDate[Jobs] = [1,10,2,4,9,8];
DueDate[Jobs] = [4,18,10,14,14,18];
Cost[Jobs,Machines] =[   //  m1  m2  m3
                           [ 2 , 3 , 6 ], //j1
                           [ 7 , 8 , 11], //j2
                           [ 6 , 5 , 7 ], //j3
                           [ 10, 12, 12], //j4
                           [ 7 , 7 , 6 ], //j5
                           [ 12, 5 , 6 ], //j6
                         ];

Duration[Jobs,Machines] =[   //  m1  m2  m3
                               [ 3 , 2 , 4 ], //j1
                               [ 6 , 4 , 5 ], //j2
                               [ 7 , 7 , 6 ], //j3
                               [ 5 , 8 , 7 ], //j4
                               [ 3 , 5 , 4 ], //j5
                               [ 5 , 6 , 5 ], //j6
                         ];
```

We assume that solving the master problem led to the assignment $\overline{\texttt{Assignment}} = \texttt{[1,1,2,1,3,2]}$. Now, applying σ_I as in Example 3.1 to select the relevant data for jobs assigned to, e.g., machine 2, we obtain the following data set:

```
n_jobs_2 = 2; n_machines_2 = 1; horizon_2 = 15;
ReleaseDate_2[Jobs_2] = [2,8];
DueDate_2[Jobs_2] = [10,18];
Cost_2[Jobs_2,[2..2]] = [ //  m2
                            [ 5 ], //j3
                            [ 5 ]  //j6
                        ];
Duration_2[Jobs_2,[2..2]] = [ //  m2
                                [ 7 ], //j3
                                [ 6 ]  //j6
                            ];
```

which represents the input to the second subproblem. The input to the other subproblems can be obtained in the same way.

A solution to the three subproblems is as follows:

```
// Machine 1, jobs 1, 2, 4
StartTime_1 = [1,10,4];
// Machine 2, jobs 3, 6
StartTime_2 = [2,9];
// Machine 3, job 5
StartTime_3 = [9];
```

which does not satisfy the fourth constraint. As an example, in time points 10, 11 and 12, all three machines are running.

One may argue whether the semantic criterion for checking problem separation as defined by Definition 3.3 is really necessary or not, and in particular whether simpler criteria based on syntactic aspects are equally effective. As an example, we could build the primal constraint graph (Dechter 2003, Sect. 2.1.3) of the subproblems—as defined previously—of Example 1.1, i.e., a graph with a node for each variable and an edge between any pair of variables syntactically occurring in the same constraint. A weaker notion of separability could be based on the fact that the graph we obtain is not connected and has exactly one (maximal connected) component for each machine. Anyway, as shown by the next example, the presence of *redundant* constraints shows that this is not the case.

Example 3.3 (Example 1.1, continued) Let the following constraint be added to the machine scheduling problem specification:

```
/* 5. If machines are less than jobs, then at least two jobs start
at different time points. (card() returns the cardinality of a set)*/
n_machines < card(Jobs) =>
            sum(i,j in Jobs:i<>j)(StartTime[i]<>StartTime[j])>=2;
```

Note that constraint 3 logically implies constraint 5, hence any solution satisfying 1–3 also satisfies 5. Note also that constraint 5 involves all the secondary variables, hence its primal constraint graph is a complete graph with `card(Jobs)` nodes, one for any `StartTime` component, representing the fact that all the secondary variables are somehow mutually constrained. As a consequence, a syntactic definition based on the constraint graph would fail to recognize separability, while Definition 3.3 does not.

We now show the generality and applicability of Definitions 3.1, 3.2 and 3.3 by exhibiting a problem specification whose resulting slave problem separates. The specification refers to the *2DHP-Protein Folding* problem, and, to the best of our knowledge, its Benders decomposition has not been considered in the literature.

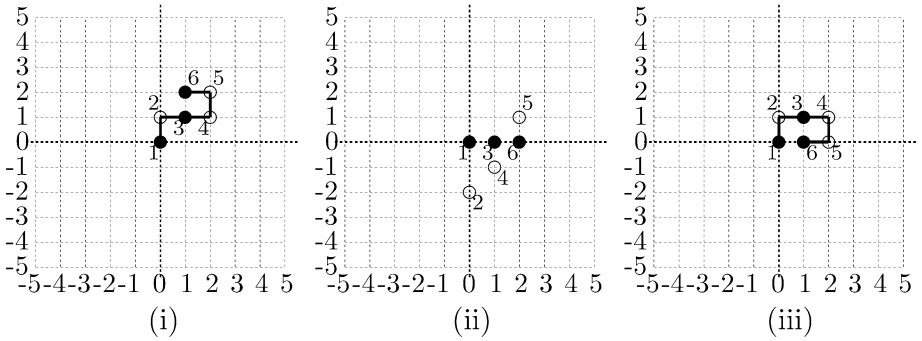

Fig. 1 Configurations of string *HPHPPH* (*solid* and *empty circles* correspond, respectively, to H and P amino-acids). (**i**) A 2DHP-PF-feasible configuration; (**ii**) A solution to the master problem, not 2DHP-PF-feasible; (**iii**) A solution to the master problem, 2DHP-PF optimal

Example 3.4 (2DHP-protein folding (Lau and Dill 1989)) 2DHP-protein folding (2DHP-PF) is a simplified version of an important problem in computational biology, which consists in finding the minimal energy spatial conformation of a (simplified) protein—i.e., a sequence of *hydrophobic*, H, and *polar* (or *hydrophilic*), P, amino-acids.

The goal is to arrange the protein on a bi-dimensional (2D) discrete grid, centering the first amino-acid, avoiding overlapping and maximizing the number of "contacts", defined as pairs of non-sequential, adjacent amino-acids H.

2DHP-PF can be modeled by representing the input protein as a string of characters in $\{H, P\}$ and the output as a sequence of folding moves (namely North, East, South, West), maximizing the number of contacts. Figure 1(i) shows a 1-contact (pair $\langle 3, 6\rangle$) conformation of sequence *HPHPPH*, obtained by applying moves: North, East, East, North, West.

The following code fragment shows an OPL model for the 2DHP-PF problem with input sequence *HPHPPH*:

```
int+ n = 6; // INPUT: Length of the string
enum Aminoacid {H,P};
range Pos [1..n]; range PosButLast [1..n-1];
Aminoacid seq[Pos] = [H,P,H,P,P,H]; // INPUT: The amino-acids string
enum Dir {N,E,S,W}; range Coord [-(n-1)..n-1];
// OUTPUT:
var Dir Moves[PosButLast]; // The folding moves sequence
// The amino-acids position (absolute coordinates):
var Coord X[Pos]; var Coord Y[Pos];
// OBJECTIVE FUNCTION:
maximize
   sum(p1,p2 in Pos: p2 > p1+1 & seq[p1] = seq[p2] = H)
    (abs(X[p1]-X[p2])+abs(Y[p1]-Y[p2])=1)
// CONSTRAINTS:
subject to {
   X[1] = 0;  Y[1] = 0; // 1.Position of first amino-acid
   forall (t in PosButLast) { // 2.Channelling constraints for positions
       Moves[t] = N => (X[t+1] = X[t]     & Y[t+1] = Y[t] + 1);
       Moves[t] = S => (X[t+1] = X[t]     & Y[t+1] = Y[t] - 1);
       Moves[t] = E => (X[t+1] = X[t] + 1 & Y[t+1] = Y[t]);
       Moves[t] = W => (X[t+1] = X[t] - 1 & Y[t+1] = Y[t]);
   };
   // 3.Avoid overlapping
   forall (p1, p2 in Pos: p1 <> p2)(X[p1]<>X[p2]\/Y[p1]<>Y[p2]);
};
```

Referring to the general form of problems (1) and its corresponding decomposition schema (2–3), we observe that:

- the objective function is defined only over arrays `X[]` and `Y[]`. Therefore, they are considered as primary variables while `Moves[]` are treated as secondary ones;
- $\alpha(\boldsymbol{s})$ is identically true in 2DHP-PF, as no constraint involving only secondary variables exists;
- $\gamma(\boldsymbol{p})$ corresponds to the conjunction of 2DHP-PF constraints 1 and 3, as they are defined only on primary variables;
- $\beta(\boldsymbol{p}, \boldsymbol{s})$ corresponds to the 2DHP-PF channelling constraint 2, which involves both primary and secondary variables.

Consequently, the master problem (*MP*) consists in placing the amino-acids in a maximal-contacts configuration, centering the first one (constraint 1) and avoiding overlapping (constraint 3), while the slave problem (*SP*) amounts to check whether a *MP* solution is a "legal" chain, i.e., if subsequent amino-acids are placed according to some legal move (constraint 2). Figure 1(ii) shows a 2-contacts, *MP*-optimal solution which does not satisfy such check. In Fig. 1(iii) a feasible and optimal solution to the 2DHP-PF instance is shown.

In order to show how Definition 3.1 applies to the obtained *SP*, we must first define an integer q and two functions σ_I and σ_V:

- $q =$ `n-1`;
- let σ_I be the function which, given an integer $i \in [1 \dots q]$ and two arrays of constants, $\overline{\mathtt{X}}$ and $\overline{\mathtt{Y}}$, returns a triple $\sigma_I(\overline{\mathtt{X}}, \overline{\mathtt{Y}}, i) =< \mathtt{X'}, \mathtt{Y'}, \mathtt{n'} >$, where $\mathtt{X'} =< \overline{\mathtt{X}}\mathtt{[i]}, \overline{\mathtt{X}}\mathtt{[i+1]} >$ and $\mathtt{Y'} =< \overline{\mathtt{Y}}\mathtt{[i]}, \overline{\mathtt{Y}}\mathtt{[i+1]} >$ are two bi-dimensional arrays and `n'=2` is an integer value representing the length of `X'` and `Y'`;
- let σ_V be the function which, given an integer $i \in [1 \dots q]$, returns the (singleton) set of variables $\sigma_V(i) = \{\mathtt{Moves[i]}\}$.

With such choice and for each $i \in [1, \dots, q]$, a (constraint satisfaction) subproblem ψ_i, including only constraint 2, is defined over the single variable in $\sigma_V(i)$. Its input instances are obtained by applying function $\sigma_I(\mathtt{X}, \mathtt{Y}, i)$ to *SP* input arrays $\overline{\mathtt{X}}$ and $\overline{\mathtt{Y}}$. ψ_i consists in finding, if any, a move such that the (i)-th and $(i+1)$-th (with respect to their order in `seq`) amino-acids are assigned consistent positions. For example, considering the positions assignment given in Fig. 1ii, where $\overline{\mathtt{X}}$=`[0,0,1,1,2,2]` and $\overline{\mathtt{Y}}$=`[0,-2,0,-1,1,0]`, if we select, e.g., $i = 3$, the subproblem ψ_3 over variables {`Moves[3]`} is defined over the input instance $< \overline{\mathtt{X}} =$ `[1,1]`, $\overline{\mathtt{Y}} =$ `[0,-1]`, `2` $>$. It can be seen that `Moves[3]=S` is a valid solution.

For any *SP* input instance, we can define q subproblems, as described above, and easily compose their respective solutions, using Definition 3.2, into an *SP* variables assignment. To do this, it is sufficient to assign, for each $i \in [1, \dots, q]$, *SP* variable `Moves[i]` the value of an i-th subproblem (ψ_i) solution. Note that, indeed, if a solution to ψ_i exists then it is unique.

Recalling the semantics of *SP* and its subproblems, we can see that, for any *SP* instance, the *SP* solutions set coincides with the set of assignments obtained by composing the *SP* subproblems solutions. In fact, it is sufficient to interpret the i-th component of any *SP* solution, namely `Moves[i]` ($i \in [1, \dots, q]$), as a solution to the i-th subproblem and, vice versa, considering each i-th subproblem solution as the i-th component of an *SP* solution. Such observation leads us to conclude that Definition 3.3 applies, i.e., that 2DHP-PF is (σ_I, σ_V)-separated.

Finally, we point out that although the provided model might not appear the most compact one, the introduction of auxiliary variables, such as `Moves`, is quite common in CSP

modeling, as it eases the writing task for a human modeler and, in some cases, yields performance improvement (Cadoli et al. 2006; Smith et al. 2000). Specifically, in this case, we showed a formulation where one may take advantage of Benders decomposition with separable subproblem.

4 Characterization of separation

Definition 3.3 gives a semantic notion of separation of a problem into subproblems. A practical difficulty is that it is not obvious how to use it for proving separation, since we would have to consider all possible instances, solve the problem and the candidate subproblems, and check that their solutions coincide.

The following theorem shows that in principle it is not necessary to do that, and reduces the problem of checking separation to the problem of equivalence of two logical formulae.

Theorem 4.1 *Given a problem $\psi(\boldsymbol{R})$, an integer q, two functions σ_I, σ_V, and q problems $\psi_1, \ldots, \psi_q$ as in Definition 3.1, ψ is (σ_I, σ_V)-separated into $\psi_1, \ldots, \psi_q$ iff the following formula is a tautology*

$$\psi \equiv \bigwedge_{i=1}^{q} \psi_i. \tag{6}$$

Proof (Only if part) First of all, we note that (6) is a tautology if and only if the following containment relations hold:

$$\forall \boldsymbol{R} \quad SOL(\psi(\boldsymbol{R})) \subseteq SOL\left(\bigwedge_{i=1}^{q} \psi_i(\boldsymbol{R})\right), \tag{7}$$

$$\forall \boldsymbol{R} \quad SOL(\psi(\boldsymbol{R})) \supseteq SOL\left(\bigwedge_{i=1}^{q} \psi_i(\boldsymbol{R})\right). \tag{8}$$

We now prove them separately, starting from the former. By hypothesis, the following q problems (σ_I,σ_V)-separate ψ:

$$\psi_i(\boldsymbol{R}) = \exists \boldsymbol{R}_i, F_i : \sigma_V(D, i) \to C \quad \text{s.t.} \quad \boldsymbol{R}_i = \sigma_I(\boldsymbol{R}, i) \wedge \phi(\boldsymbol{R}_i, F_i) \quad (i = 1, \ldots, q). \tag{9}$$

Now, given an instance $\boldsymbol{R} \in \mathcal{R}$, if F is a solution to $\psi(\boldsymbol{R})$ then any restriction $F|_{D_i}$ to $D_i = \sigma_V(D, i)$ is a solution to $\psi_i(\boldsymbol{R})$ for each $i = 1, \ldots, q$. In fact, separation implies that (cf. Definitions 3.2 and 3.3):

$$SOL(\psi(\boldsymbol{R})) = \{F : D \to C \text{ s.t. } \forall i = 1, \ldots, q \ F|_{D_i} \in SOL(\psi_i(\boldsymbol{R}))\}.$$

Hence, for each instance $\boldsymbol{R}$, if F solves $\psi(\boldsymbol{R})$ then it solves the q problems $\psi_i(\boldsymbol{R})$ or, equivalently, F is a solution to the problem

$$\bigwedge_{i=1}^{q} \psi_i(\boldsymbol{R})$$

and then relation (7) holds.

We now turn to the inverse containment (8). To this end, consider a solution $G : D \to C$ to the problem $\bigwedge_{i=1}^{q} \psi_i(\boldsymbol{R})$ for a generic instance $\boldsymbol{R}$. By definition, G solves all of the $\psi_i(\boldsymbol{R})$ problems and, recalling the form (9) of ψ_i, it follows that $G|_{\sigma_V(D,i)}$ solves $\psi_i(\boldsymbol{R})$. In other words, G is such that:

$$\forall i = 1, \ldots, q \quad G|_{\sigma_V(D,i)} \in SOL(\psi_i(\boldsymbol{R})).$$

The separability hypothesis implies that $G \in SOL(\psi(\boldsymbol{R}))$, hence relation (8) holds.

(If part) Assuming (6) is a tautology, $F : D \to C$ is a solution to $\psi(\boldsymbol{R})$, for any $\boldsymbol{R}$, if and only if F solves the problem $\bigwedge_{i=1}^{q} \psi_i(\boldsymbol{R})$ or, equivalently, the q problems $\psi_i(\boldsymbol{R})$ $(i = 1, \ldots, q)$. Consequently, any solution F to $\psi(\boldsymbol{R})$ is such that $\forall i = 1, \ldots, q \ F|_{D_i} \in SOL(\psi_i(\boldsymbol{R}))$ and vice versa. Hence, for any $\boldsymbol{R}$,

$$SOL(\psi(\boldsymbol{R})) = \{F : D \to C \text{ s.t. } \forall i = 1, \ldots, q \ F|_{D_i} \in SOL(\psi_i(\boldsymbol{R}))\}. \qquad \square$$

Theorem 4.1 calls for an equivalence check among logical formulae. This task is non decidable even for first-order formulae (Börger et al. 1997), and actually this lower bound applies also to this case, as shown by the next theorem. However, before proving formally such result we need to introduce a new class of problems, by means of a specialization of Definition 3.3.

Definition 4.1 (Strongly non-separated problem) Given a problem of the form (5) and its q subproblems $\psi_i(\boldsymbol{R})$ as in Definition 3.1, ψ is *strongly not* (σ_I, σ_V)*-separated into the* q *problems* $\psi_1, \ldots, \psi_q$ iff

$$\forall \boldsymbol{R} \in \mathcal{R} \quad \bowtie_{i=1}^{q} SOL(\psi_i(\boldsymbol{R})) \neq SOL(\psi(\boldsymbol{R})). \tag{10}$$

Note that a strongly non-separated problem is not just a problem which is not separated by (σ_I, σ_V): for the latter it suffices *one instance* of input data $\boldsymbol{R} \in \mathcal{R}$ such that equality between sets of solutions $\bowtie_{i=1}^{q} SOL(\psi_i(\boldsymbol{R}))$ and $SOL(\psi(\boldsymbol{R}))$ does not hold. On the other hand, for strongly non-separated problems equality between sets of solutions does not hold for *all instances* of input data.

It is easy to see that strongly non-separated problems do exist.

Example 4.1 Consider the problem P obtained by:

- same search space as in Example 1.1;
- constraint 3 of Example 1.1;
- a constraint (similar to constraint 4 of Example 3.2), stating that there is no more than one machine running at the same time;
- a constraint stating that there are exactly two machines.

Note that no constraint involves either start or due date and, as a consequence, job start times can be assigned arbitrarily.

Let (σ_I, σ_V) be like in Example 3.1. (σ_I, σ_V) do not describe a separation of P, because for each instance $\boldsymbol{R}$ and each solution $\boldsymbol{S}$ of P, we can always obtain a solution of the subproblems in which machines run in parallel. In other words, relation (10) holds.

As anticipated above, the notion of strongly non-separated problems is introduced with the aim of proving the next theorem. To this end, showing that at least one strongly non-separated problem exists is fundamental and it is exactly the purpose of Example 4.1.

Theorem 4.2 *Given a problem $\psi(\boldsymbol{R})$, an integer q, two functions σ_I, σ_V, and q problems $\psi_1, \ldots, \psi_q$ as in Definition 3.1, it is not decidable to check whether ψ is (σ_I, σ_V)-separated or not.*

Proof Consider a problem $\psi(\boldsymbol{R})$ of the form:

$$\exists F : D \to C \quad \text{s.t.} \quad \eta(\boldsymbol{R}, F) \wedge (\xi(\boldsymbol{R}) \to \pi(\boldsymbol{R}, F)) \tag{11}$$

where $\xi(\boldsymbol{R})$ is a first-order formula in which only input data symbols occur, and that represents a "filter" on input data. Assume that the problem

$$\psi_1(\boldsymbol{R}) = \exists F : D \to C \quad \text{s.t.} \quad \eta(\boldsymbol{R}, F) \tag{12}$$

is (σ_I, σ_V)-separated, and that the problem

$$\psi_2(\boldsymbol{R}) = \exists F : D \to C \quad \text{s.t.} \quad \pi(\boldsymbol{R}, F),$$

is strongly not (σ_I, σ_V)-separated.

As an example of $\psi_1(\boldsymbol{R})$ just take the third constraint from Example 1.1. As an example of $\psi_2(\boldsymbol{R})$ problem, just take the problem described in Example 4.1.

Of course the problem

$$\exists F : D \to C \quad \text{s.t.} \quad \eta(\boldsymbol{R}, F) \wedge \pi(\boldsymbol{R}, F)$$

is not (σ_I, σ_V)-separated, being the conjunction of two constraints, the former being (σ_I, σ_V)-separated and the latter being strongly not (σ_I, σ_V)-separated.

Now note the role played by formula $\xi(\boldsymbol{R})$ in problem (11). If $\xi(\boldsymbol{R})$ is identically false, then problem (11) coincides with problem (12), and it is (σ_I, σ_V)-separated. If $\xi(\boldsymbol{R})$ is not identically false, then let R be an instance of input data such that $\xi(R)$ is satisfiable. According to Definition 4.1, R proves that problem (11) is not (σ_I, σ_V)-separated. Summing up, problem (11) is (σ_I, σ_V)-separated if and only if first-order formula $\xi(\boldsymbol{R})$ is identically false, which is not decidable (Börger et al. 1997). □

The undecidability of the problem of checking separation puts severe restrictions on the possibility of mechanizing the process of finding, or at least validating, Benders decompositions. Nevertheless, in what follows we show that current Automated Theorem Provers (ATP) technology can be effectively used for checking the separation property, that is, ultimately, for showing that the obtained Benders cuts rule out several (instead of one) trial values from the Master problem's search space.

In particular, in the following example we show a feasibility problem whose separation, according to Theorem 4.1, can be checked by a first-order ATP.

Example 4.2 (All colors monochromatic 2-cycle existence (M2CE)) Given a directed colored graph, M2CE consists in deciding whether, for each color, a length-two cycle of monochromatic nodes exists. This problem can be separated considering colors (and the induced monochromatic subgraphs) separately.

In what follows we show:

1. a representation of M2CE in the format of formula (5), as well as functions σ_I and σ_V characterizing a decomposition into subproblems via Definition 3.1;

2. that (σ_I, σ_V)-separation of M2CE through Theorem 4.1 can be automatically proven.

As for step 1, assuming that the input instance is represented through predicate symbols $edge/2$ ($edge(X, Y)$: there is an edge between nodes X and Y) and $col/2$ ($col(X, C)$: node X has color C), M2CE can be formally described as a first-order formula as follows:

$$\forall C\ \exists XY \quad X \neq Y \wedge col(X, C) \wedge col(Y, C) \wedge edge(X, Y) \wedge edge(Y, X). \tag{13}$$

The second order representation in the format of (5) can be obtained by introducing a new, existentially quantified, predicate $gc/1$ ($gc(C)$: C is a "good color", i.e., a color for which a length-two cycle exists) as follows

$$\exists gc\ (\forall C\ gc(C))$$
$$\wedge (gc(C) \leftrightarrow \exists XY\ X \neq Y \wedge col(X, C) \wedge col(Y, C) \wedge edge(X, Y) \wedge edge(Y, X)). \tag{14}$$

gc is a function with the colors as domain and the Booleans as range. As for the integer q and the functions σ_I and σ_V (cf. Definition 3.1), they are as follows:

- q is the number of colors;
- σ_I selects, for each color $i \in 1..q$, the nodes colored with i and their edges from predicate symbols *edge* and *col*;
- σ_V selects, for each color $i \in 1..q$, color i from predicate symbol gc.

As for step 2, we assume that there are exactly two colors, and prove (σ_I, σ_V)-separation of M2CE. The proof is obtained by proving that the formula

$$\neg\left(\psi \equiv \bigwedge_{i=1}^{2} \psi_i\right) \tag{15}$$

is unsatisfiable, i.e., that its negation is a tautology, where ψ is formula (13), and ψ_i ($1 \leq i \leq 2$) are obtained from ψ, σ_I, and σ_V according to Definition 3.1.

For doing this, we feed OTTER[2] (Quaife 1992; Wos 1996), a well-known resolution-based first-order ATP, with the following file:

```
%%% types and disjointness
all X Y (edge(X,Y) -> node(X) & node(Y)).
all X C (col(X,C) -> node(X) & color(C)).
all X   (node(X) -> - color(X)).
%%% col is a function
all X C1 C2 (col(X,C1) & col(X,C2) -> C1 = C2).
%%% there are 2 colors: c1 and c2
2col <-> (c1 != c2 & (all X C (col(X,C) <-> C = c1 | C = c2))).
%%% sigma_I
all X (col_1(X) <-> col(X,c1)).
all X (col_2(X) <-> col(X,c2)).
all X Y (edge_1(X,Y) <-> edge(X,Y) & col(X,c1) & col(Y,c1)).
all X Y (edge_2(X,Y) <-> edge(X,Y) & col(X,c2) & col(Y,c2)).
%%% Psi
e2c <->
   (all C (exists X Y (X != Y & col(X,C) & col(Y,C) & edge(X,Y) & edge(Y,X)))).
%%% Psi_1
e2c_1 <->
```

[2] http://www-unix.mcs.anl.gov/AR/otter

```
(all C
%%% C does not occur, but we quantify it according to the rules
%%% for building Psi_1
  (exists X Y (X != Y & col_1(X) & col_1(Y) & edge_1(X,Y) & edge_1(Y,X)))
).
%%% Psi_2
e2c_2 <->
(all C
%%% idem
  (exists X Y (X != Y & col_2(X) & col_2(Y) & edge_2(X,Y) & edge_2(Y,X)))
).
separable <-> (2col -> ((e2c_1 & e2c_2) <-> e2c)).
%%% we require a refutation
- separable.
```

OTTER is able to prove unsatisfiability of the above formula (in full automatic mode) in less than one second.

5 Conclusions

In this paper we have analyzed the notion of separation of problems. This is a concept interesting *per se*, and finds an immediate application in the context of Benders decompositions. In fact, it is well-known (Hooker and Ottosson 2003; Hooker 2000; Cambazard and Jussien 2005) that when the slave problem is formulated using several subproblems exhibiting strong intra-relationships and weak inter-relationships such decompositions are effective.

In the literature, informal notions of separation of subproblems are typically used, but in this paper we have shown that it is not easy at all to come up with a clear syntactical definition of separability. Examples of Sects. 1 and 3 show that formulations of a problem which look similar from the syntactical point of view may or may not be separable. A precise, semantic definition of separation has been provided in Sect. 3, which has been characterized both from the logical and from the computational points of view in Sect. 4.

In particular, we have shown that separation can be reduced to checking equivalence of second-order logic formulae (Theorem 4.1), and that the problem of checking whether a given selection of input data corresponds to a separation or not is not decidable (Theorem 4.2).

Moreover, we have shown that there are special cases where Theorem 4.1 calls for first-order, instead of second-order, equivalence and an automated proof of separation can be obtained by means of an Automated Theorem Prover (Example 4.2). Of course, the undecidability result is valid also in such cases, the prover giving no guarantee of termination. However, in those cases where it terminates, the answer can be exploited to state whether separation holds.

As for the current work, since the notion of separation into subproblems seems to be related to the concept of *database integration*, especially in the context of different information sources, cf. e.g., Lenzerini (2002), we plan to extend our definitions in the traditional database context.

Acknowledgements This paper is in memory of Marco Cadoli.

The authors would like to thank the anonymous reviewers for their useful comments, which have been very helpful for improving the quality of the paper.

References

Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, *4*, 238–252.

Börger, E., Gräedel, E., & Gurevich, Y. (1997). *The classical decision problem*. *Perspectives in mathematical logic*. Berlin: Springer.

Cadoli, M., Mancini, T., Micaletto, D., & Patrizi, F. (2006). Evaluating ASP and commercial solvers on the CSPLib. In *Proceedings of the seventeenth European conference on artificial intelligence (ECAI 2006)*.

Cambazard, H., & Jussien, N. (2005). Integrating Benders decomposition within constraint programming. In *Lecture notes in artificial intelligence: Vol. 3709*. *Proceedings of the eleventh international conference on principles and practice of constraint programming (CP 2005)* (pp. 752–756). Berlin: Springer.

Dechter, R. (2003). *Constraint processing*. San Mateo: Morgan Kaufmann.

Fagin, R. (1974). Generalized first-order spectra and polynomial-time recognizable sets. In R. M. Karp (Ed.), *Complexity of computation* (pp. 43–74). Providence: Am. Math. Soc.

Hooker, J. (2000). *Logic-based methods for optimization: combining optimization and constraint satisfaction* (Chap. 19, pp. 389–422). New York: Wiley.

Hooker, J. N., & Ottosson, G. (2003). Logic-based Benders decomposition. *Mathematical Programming*, *96*, 33–60.

Jain, V., & Grossmann, I. E. (2001). Algorithms for hybrid MILP/CP models for a class of optimization problems. *INFORMS Journal on Computing*, *13*, 258–276.

Lau, K. F., & Dill, K. A. (1989). A lattice statistical mechanics model of the conformational and sequence spaces of proteins. *Macromolecules*, *22*, 3986–3997.

Lenzerini, M. (2002). Data integration: A theoretical perspective. In *Proceedings of the twentyfirst ACM SIGACT SIGMOD SIGART symposium on principles of database systems (PODS 2002)* (pp. 233–246).

Papadimitriou, C. H. (1994). *Computational complexity*. Reading, MA: Addison Wesley.

Puget, J. F. (1998). A fast algorithm for the bound consistency of alldiff constraints. In *Proceedings of the fifteenth national conference on artificial intelligence (AAAI'98)* (pp. 359–366).

Quaife, A. (1992). *Automated development of fundamental mathematical theories*. Dordrecht: Kluwer Academic.

Smith, B. M., Stergiou, K., & Walsh, T. (2000). Using auxiliary variables and implied constraints to model non-binary problems. In *AAAI/IAAI* (pp. 182–187).

Van Hentenryck, P. (1999). *The OPL optimization programming language*. Cambridge: MIT Press.

Walsh, T. (2001). Permutation problems and channelling constraints. In *Lecture notes in computer science: Vol. 2250*. *Proceedings of the eighth international conference on logic for programming, artificial intelligence and reasoning (LPAR 2001)* (pp. 377–391). Berlin: Springer.

Wos, L. (1996). *The automation of reasoning: An experimenter notebook with OTTER tutorial*. San Diego: Academic Press.