On the separability of subproblems in Benders decompositions

Marco Cadoli and Fabio Patrizi

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza", Italy
{cadoli|patrizi}@dis.uniroma1.it

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 *subproblem*. Its effectiveness relies on the possibility of synthethising *Benders cuts* (or nogoods) that rule out not only one, but a large class of trial values for the master problem. In turns, this depends on the possibility of *separating* the subproblem into several subproblems, i.e., problems exhibiting strong intra-relationships and weak inter-relationships. The notion of *separation* is typically given informally, or relying on syntactical aspects. This paper formally addresses the notion of separability of the subproblem by giving a semantical definition and exploring it from the computational point of view. Several examples of separable problems are provided, including some proving that a semantical 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 problem of checking separability.

1 Introduction and Motivations

Benders decomposition [1] 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 defining a *subproblem* over secondary variables given a trial value for primary variables. Every unsuccessful attempt to solve the subproblem is recorded as a *Benders cut* (or *nogood*) and added to the master problem, until an optimal solution is found, or the problem is proven to be unfeasible.

Two important factors that make the above procedure effective are: 1) the possibility of using different technologies for solving the master and the subproblem, e.g., ILP and CP, respectively, [9, 10], and 2) the possibility of synthethising 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 subproblem, 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 [9, 8, 4] that, if the subproblem is separable, then it is possible to design a Benders cut that excludes several instantiations of the primary variables, or, in other words, a nogood which is a partial, and not a total, assignment to the primary variables.

Therefore, the ability of recognizing separability of the subproblem is crucial for the efficiency of a Benders decomposition.

Let us introduce our running example, taken from [9, 8], which refers to a *machine scheduling* problem.

Example 1 (Machine Scheduling Problem [9, 8]). 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 [15], one possible model is as follows:

```
// INPUT DESCRIPTION
{int+} Jobs=...;                   //The set of jobs to be scheduled
int+ horizon=...;                  //Max start time point for jobs
int+ n_machines=...;               //The 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 [9] selects `Assignment` and `StartTime` as primary and secondary variables, respectively. Moreover, it defines the master problem as the minimization problem on no constraints, nd the subproblem as the decision problem on constraint 1, 2, and 3 (for a given optimal instantiation of $\overline{\texttt{Assignment}}$ and ignoring the objective function). A given optimal instantiation $\overline{\texttt{Assignment}}$ which is unfeasible for the subproblem 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 unfeasible.

This is a "raw" version of the decomposition, which ignores the fact that the subproblem is *separable* wrt the machines. As an example, if we have three machines, we can consider three separate subproblems, one for each of them. If $\overline{\texttt{Assignment}}$ = [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 the latter kind rule out a whole set of assignments (not just one) to the primary variables, and can be added for each unfeasible subproblem. This ultimately results in a more efficient decomposition. □

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.

Example 2 (Example 1, continued). Since constraint 3 is universally quantified wrt the machines "`forall (m in Machines)`", we can claim its separability just relying on an intuitive argument. The methodological problem is that this syntax-based argument is heavily dependent on the way the problem is formulated (cf. also forthcoming Example 6). 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;
```

Since constraint 3' is not universally quantified wrt the machines, it is less clear that it separates. □

It is everyday experience that different formulations, all of them being intuitive, can be done for a problem, sometimes in the hope of having more performant models.

Example 3 (Example 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 [14], especially in connection with "channelling constraints" [16].

```
range MachinesPlus[-card(Jobs)..n_machines];//negative numbers: irrelevant
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 wrt the machines, but it nevertheless separates. □

In this paper we investigate the possibility of automating the process of checking subproblem separability in the context of Benders decompositions. In particular, given a problem and applying to it a given Benders decomposition schema which leads to a constraint satisfaction subproblem [8], our goal is to state the conditions, if any, that make the subproblem separable. To this end, we first address the notion of subproblem separability by giving a semantical 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 undecidable.

The exposition is structured as follows. In Section 2 we recall the definition of Benders decomposition, in Section 3 a formal definition of separation is given, while in Section 4 we show semantical and computational characterizations. Finally, Section 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 $P = C_1^p \times \ldots \times C_n^p$ and $S = C_1^s \times \ldots \times C_m^s$, in this paper we consider problems of the form:

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

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 [9, 4] generalizations of the above problem in which, e.g., variables from $\boldsymbol{s}$ may occur in the objective function, are studied. According to [9], such problems can be solved by applying a "logic-based" Benders Decomposition scheme that gives raise to the following problems:

$$MP^k : \begin{cases} \min\{f(\boldsymbol{p})\} & o.f. \\ s.t. & \\ \gamma(\boldsymbol{p}) & c2 \\ CUT_{\boldsymbol{p}^i}(\boldsymbol{p}) & \\ (i = 1, 2, \ldots k-1) & \textit{Benders cuts} \\ \boldsymbol{p} \in P & p.v.d. \end{cases} \qquad SP : \begin{cases} \alpha(\boldsymbol{s}) & c1 \\ \beta(\overline{\boldsymbol{p}}, \boldsymbol{s}) & c3 \\ \boldsymbol{s} \in S & s.v.d. \end{cases} \tag{2}$$

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 satifying 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.

Subproblem (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, $\boldsymbol{p}$ is `Assignment`, $\boldsymbol{s}$ is `StartTime`, $\alpha(\boldsymbol{s})$ is constraint 1, $\beta(\boldsymbol{p}, \boldsymbol{s})$ is the conjunction of constraints 2 and 3, 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

$$CUT_{\boldsymbol{p}^k}(\boldsymbol{p}) \doteq (\boldsymbol{p} \neq \boldsymbol{p}^k), \tag{3}$$

where $\boldsymbol{p}^k$ is the solution to MP^k, is sound. The problem with (3) is that an unacceptably large number of cuts may be added to the Master problem, and this may reflect in inefficiency (cf. Example 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 Section 1, we need to clarify the role played by the selection of relevant input data. Referring to Example 1, every choice of the machine induces a selection of the release and due dates, costs, and durations. As an example, if $\overline{\texttt{Assignment}}$ = `[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 $\sigma_1 : \mathcal{R} \times [1, q] \rightarrow \mathcal{R}$ that *selects* the input data relevant for the i-th subproblem $(1 \leq i \leq q)$.

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_2 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 problems with the following notation

$$\psi(\boldsymbol{R}) = \exists F : D \rightarrow C \ s.t. \ \phi(\boldsymbol{R}, F), \tag{4}$$

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 domain and the codomain of the assignment, respectively, and $\phi(\boldsymbol{R}, F)$ is a representation of the constraints. We prefer the above notation over the notation as in (1) or (2) 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 (4) can represent every problem in the complexity class NP [7, 13]. 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 in Example 1 corresponds to the existentially quantified function F in (4).

Definition 1 (Subproblems). *Given a problem ψ of the form (4), an integer $q \geq 1$ and two functions $\sigma_1 : \mathcal{R} \times [1, q] \to \mathcal{R}$ and $\sigma_2 : [1, q] \to 2^D$ such that $\{D_1, \ldots, D_q\}$ ($D_i = \sigma_2(i)$) is a partition of D, the following q problems are defined as the* subproblems of ψ wrt σ_1 and σ_2*:*

$$\psi_i(\boldsymbol{R}) = \exists \boldsymbol{R}_i, F_i : D_i \to C \ s.t. \ \boldsymbol{R}_i = \sigma_1(\boldsymbol{R}, i) \land \phi(\boldsymbol{R}_i, F_i), \qquad (i = 1, \ldots, q).$$

Definition 1 can be used to obtain the subproblems in a syntactical way, by means of a symbolic manipulation of the problem. To see intuitively how the subproblems are obtained, we resort again to our running example.

Example 4 (Example 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_1 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_1 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]);
//OUPUT:
{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 1 where the $\boldsymbol{R}_i$ 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_2 takes a machine `i` and the variables, i.e., array `StartTime`, and gives a new array of variables `StartTime_i`. The representation of σ_2 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, 2, and 3. $\Box$

Given an instance $\boldsymbol{R}$ of a problem of the form (4), 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 2 (Composition of solutions). *Given a problem $\psi(\boldsymbol{R})$ and its q subproblems $\psi_i(\boldsymbol{R})$ as in Definition 1, we define the* composition ($\bowtie$) of the solutions $SOL(\psi_i(\boldsymbol{R}))$ *of the subproblems as follows:*

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

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 *semantical*, i.e., based on the respective solutions. The following definition tells us that a problem is separated by (σ_1, σ_2) if its solutions can be obtained just by composing the solutions of its subproblems.

Definition 3 (Separation). *Given a problem of the form (4) and its q subproblems $\psi_i(\boldsymbol{R})$ as in Definition 1, ψ is (σ_1, σ_2)-separated into the q problems $\psi_1, \ldots, \psi_q$ iff*

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

Referring again to the subproblem in the three versions of Examples 1, 2 and 3, it is possible to see that it is (σ_1, σ_2)-separated into q problems according to Definition 3, where q is the number of machines.

Of course, not all problems are separable, as shown by the next example.

Example 5. We add to the constraints of Example 1 a further constraint which avoids more than 2 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 subproblem, and the Master problem is unchanged. With functions σ_1 and σ_2 defined as in Example 4, it is possible to see that the current version of the subproblem is not (σ_1, σ_2)-separated. We do that by 1) exhibiting an instance, 2) solving separately the subproblems obtained applying Definition 1, 3) composing their solutions according to Definition 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 $\overline{\texttt{Assignment}}$ = `[1,1,2,1,3,2]`. Now, applying σ_1 as in Example 4 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 2nd separated 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 semantical criterion for checking problem separation as defined by Definition 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 [6] of the subproblems –as defined previously– of Example 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 has one component for each machine. Anyway, as shown by the next example, a problem with *redundant* constraints arises.

Example 6 (Example 1, continued). Let the following constraint be added to the machine scheduling problem specification:

```
/* 4. 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 4, hence any solution satisfying 1-3 also satisfies 4. Note also that constraint 4 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 does not. □

Definition 3 is clarified also by the following example.

Example 7 (Protein Folding). [11] This problem specification models a simplified version of an important problem in computational biology which consists in finding the spatial conformation of a protein (i.e., a sequence of amino-acids) with minimal energy.

The simplifications with respect to the real problem are twofold: firstly, the 20-letter alphabet of amino-acids is reduced to a two-letter alphabet, namely H and P. H represents *hydrophobic* amino-acids, whereas P represents polar or *hydrophilic* amino-acids. Secondly, the conformation of the protein is limited to a bi-dimensional discrete space. Nonetheless, these limitations have been proven to be very useful for attacking the whole protein conformation prediction protein, which is known to be NP-complete [5] and very hard to solve in practice.

In this formulation, given the sequence (of length n) of amino-acids of the protein (the so called primary structure of the protein), i.e., a sequence of length

n with elements in {H,P}, we aim to find a connected shape of this sequence on a bi-dimensional grid (whose points have coordinates in the integral range $[-(n-1),(n-1)]$, the sequence starting at $(0,0)$), which is not overlapping, and maximizes the number of "contacts", i.e., the number of non-sequential pairs of H amino-acids for which the Euclidean distance of the positions is 1 (the overall energy of the protein is defined as the opposite of the number of contacts).

Protein Folding can be modeled as a planning problem, whose input is a representation of the protein and the output is a sequence of moves that maximizes the number of contacts by folding it.

More specifically, a protein is described by a sequence $\boldsymbol{s} \in \{0,1\}^n$ (1 and 0 representing, respectively, H and P), that can be arranged on the grid by means of four moves: ***u***p, ***d***own, ***l***eft, ***r***ight, each of which sets the position of an amino-acid wrt its predecessor. The first element is always placed in the center of the grid and the sequence cannot cross itself. Figure 1 shows a 1-contacts configuration for a sequence $\boldsymbol{s} =< 1,0,1,0,0,1 >$ of length 6, obtained by applying, in the reported order, the moves $\boldsymbol{u}$ $\boldsymbol{r}$ $\boldsymbol{r}$ $\boldsymbol{u}$ $\boldsymbol{l}$.

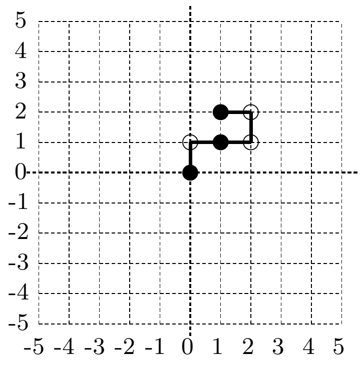

Fig. 1. An instance of the Protein folding problem

More formally, given:

- an array $\overline{\boldsymbol{s}} \in \{0,1\}^n$, representing the protein;
- a set $T_{move} = [1, n-1] \in \mathbb{N}$;
- a set $T_{pos} = [1, n] \in \mathbb{N}$;
- a set $B = [-n+1, n-1] \in \mathbb{Z}$;
- a set $M = \{\boldsymbol{u} =< 0,1 >, \boldsymbol{d} =< 0,-1 >, \boldsymbol{l} =< -1,0 >, \boldsymbol{r} =< 1,0 >\} \in \mathbb{Z}^2$;
- the set of variables $V = \{m_1, \ldots, m_{n-1}, x_1, \ldots, x_n, y_1, \ldots, y_n\}$, $m_i \in M, \forall i \in T_{move}$ and $< x_i, y_i > \in B^2, \forall i \in T_{pos}$;

and defined:

- $\boldsymbol{moves} = (m_1, \ldots, m_{n-1}) \in M^{n-1}$, the array representing the sequence of moves;
- $\boldsymbol{pos} = (< x_1, y_1 >, \ldots, < x_n, y_n >) \in (B^2)^n$, the array whose i-th component represents the position of the i-th amino-acid;
- the function
$$Hits(\boldsymbol{pos}, \overline{\boldsymbol{s}}) = \sum_{\substack{t,t' \in T_{move}\ s.t. \\ t' > t+1\ \wedge\ \overline{pos_t pos_{t'}} = 1}} \overline{s_t} \cdot \overline{s_{t'}}$$
that counts the number of contacts for a particular $\boldsymbol{pos}$;

the Protein Folding Problem can be stated as follows:

$$PF : \begin{cases} \max\{Hits(\boldsymbol{pos}, \overline{\boldsymbol{s}})\} & o.f. \\ s.t. & \\ pos_1 =< 0, 0 > & start\ condition \\ \forall t \in T_{move}\ move_t = pos_{t+1} - pos_t & channelling\ constraints \\ \forall (t, t') \in T^2_{pos} \mid t > t'\ pos_t \neq pos_{t'} & no\ crossing\ constraints \end{cases}$$

In such form, it can be decomposed as in (2) by considering $\boldsymbol{pos}$ as the array of primary variables and $\boldsymbol{moves}$ that of secondary, obtaining the Master Problems MP^k_{PF} and the Subproblem SP_{PF}:

$$MP^k_{PF} : \begin{cases} \max\{Hits(\boldsymbol{pos}, \overline{\boldsymbol{s}})\} & o.f. \\ s.t. & \\ pos_1 =< 0, 0 > & start\ condition \\ \forall (t, t') \in T^2_{pos} \mid t > t'\ pos_t \neq pos_{t'} & no\ crossing\ constraints \\ CUT_{\boldsymbol{pos}^i}(\boldsymbol{pos}) & \\ (i = 1, \ldots k-1) & Benders\ cuts \end{cases}$$

$$SP_{PF} : \left\{ \forall t \in T_{move}\ move_t = \overline{pos_{t+1}} - \overline{pos_t} \quad channelling\ constraints \right.$$

The Subproblem is to find a sequence of moves such that the given positions obtained by solving the Master Problem are feasible.
Now, apply Definition 3:

- define σ_2 as the function selecting, for each $i = 1, \ldots, n-1$, the set of secondary variables $\{m_i\}$;
- define σ_1 as the function selecting, for each $i = 1, \ldots, n-1$, the pair (pos_i, pos_{i+1}) from $\boldsymbol{pos}$.

Such functions define the $n-1$ problems

$$SP^i_{PF} : \left\{ move_i = \overline{pos_{i+1}} - \overline{pos_i} \right.\ , i = 1, \ldots, n-1$$

of checking whether there exists a move such that the $(i+1)$-th amino-acid is in a position reachable from that of the i-th.
To see that SP_{PF} is (σ_1, σ_2)-separated, observe that the generic array $\boldsymbol{moves}$ is a solution to SP_{PF} for a given assignment $\overline{\boldsymbol{pos}}$ iff

$$\boldsymbol{moves} = (\overline{pos}_2 - \overline{pos}_1, \ldots, \overline{pos}_n - \overline{pos}_{n-1})$$

and that each of its components is a solution to SP_{PF}^{i}, $(i = 1, \ldots, n)$. Conversely, given $(n-1)$ solutions $moves_i$ to SP_{PF}^{i}, $(i = 1, \ldots, n)$, the array

$$\boldsymbol{moves} = (moves_1, \ldots, moves_n)$$

is a solution to SP_{PF}, as assignment $\overline{pos}_{i+1}$ of SP_{PF}^{i} is, by construction, the same as $\overline{pos}_{i}$ of problem SP_{PF}^{i+1} forall $i = 1, \ldots, n-1$. $\square$

4 Characterization of separation

Definition 3 gives a semantical notion of separation of a problem in 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 1. *Given a problem $\psi(\boldsymbol{R})$, an integer q, two functions σ_1, σ_2, and q problems $\psi_1, \ldots, \psi_q$ as in Definition 3, ψ is (σ_1, σ_2)-separated into $\psi_1, \ldots, \psi_q$ iff the following formula is a tautology*

$$\psi \equiv \bigwedge_{i=1}^{q} \psi_i. \tag{5}$$

Proof. (Only if part.) By hypothesis, the following q problems (σ_1, σ_2)-separate ψ:

$$\psi_i(\boldsymbol{R}) = \exists \boldsymbol{R}_i, F_i : \sigma_2(D, i) \rightarrow C \text{ s.t. } \boldsymbol{R}_i = \sigma_1(\boldsymbol{R}, i) \wedge \phi(\boldsymbol{R}_i, F_i), \quad (i = 1, \ldots, q). \tag{6}$$

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 $D_i = \sigma_2(D, i)$ is a solution to $\psi_i(\boldsymbol{R})$ for each $i = 1, \ldots, q$. In fact, since separation holds, it holds that

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

Hence, for any 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

$$\forall \boldsymbol{R} \ SOL(\psi(\boldsymbol{R})) \subseteq SOL(\bigwedge_{i=1}^{q} \psi_i(\boldsymbol{R})).$$

In order to prove that (5) is a tautology, we must show that also the inverse containtment holds. To this end, consider a solution $G : D \rightarrow C$ 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 (6) of ψ_i, it is straightforward that $G|_{\sigma_2(D,i)}$ solves $\psi_i(\boldsymbol{R})$. In other words, G is such that

$$\forall i = 1, 2, \ldots, q \; G|_{\sigma_2(D,i)} \in SOL(\psi_i(\boldsymbol{R})).$$

But, due to separability, it yields $G \in SOL(\psi(\boldsymbol{R}))$ and hence

$$\forall \boldsymbol{R} \; SOL(\psi(\boldsymbol{R})) \supseteq SOL(\bigwedge_{i=1}^{q} \psi_i(\boldsymbol{R})).$$

(If part.) Assuming (5) 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 viceversa. Hence, for any $\boldsymbol{R}$,

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

□

Theorem 1 calls for an equivalence check among logical formulae. This task is undecidable even for first-order formulae [2], and actually this lower bound applies also to this case, as shown by the next theorem.

Theorem 2. *Given a problem $\psi(\boldsymbol{R})$, an integer q, two functions σ_1, σ_2, and q problems $\psi_1, \ldots, \psi_q$ as in Definition 3, it is not decidable to check whether ψ is (σ_1, σ_2)-separated or not.*

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

$$\exists F : D \to C \text{ s.t. } (\forall m \in M \; \eta(\boldsymbol{R}, F, m)) \quad \wedge \quad (\xi(\boldsymbol{R}) \to \exists m \in M \; \pi(\boldsymbol{R}, F, m)), \tag{7}$$

where M is a finite domain such that $q = |M|$ and $\xi(\boldsymbol{R})$ is a first-order formula which represents a "filter" on input data. Assume that the problem

$$\exists F : D \to C \text{ s.t. } \forall m \in M \; \eta(\boldsymbol{R}, F, m), \tag{8}$$

is (σ_1, σ_2)-separated, and that the problem

$$\exists F : D \to C \text{ s.t. } \exists m \in M \; \pi(\boldsymbol{R}, F, m),$$

is not (σ_1, σ_2)-separated. As an example for the former problem just take the third constraint from Example 1. As an example of the latter problem, just take the fourth constraint from Example 5.

Of course the problem

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

is not (σ_1, σ_2)-separated, essentially being a conjunction of constraints, the former being (σ_1, σ_2)-separated and the latter being not (σ_1, σ_2)-separated.

Now note the role played by formula $\xi(\boldsymbol{R})$ in problem (7). If $\xi(\boldsymbol{R})$ is identically false, then problem (7) coincides with problem (8), and it is (σ_1, σ_2)-separated. If $\xi(\boldsymbol{R})$ is not identically false, then we can find an instance showing that problem (7) is not (σ_1, σ_2)-separated. Summing up, problem (7) is (σ_1, σ_2)-separated iff first-order formula $\xi(\boldsymbol{R})$ is identically false, which is not decidable [2]. $\square$

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, it has been shown in [3] that current Automated Theorem Provers technology can be effectively used for checking properties, such as existence of symmetries or dependence among arrays of variables, similar to separation. It is the purpose of future research to investigate on the applicability of the methodology of [3] to the separation problem.

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 that such decompositions are effective only if the subproblem is formulated using several subproblems exhibithing strong intra-relationships and weak inter-relationships.

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 1-4 show that formulations of a problem which look similar from the syntactical point of view may or may not be separable. A precise, semantical definition of separation has been provided, which has been characterized both from the logical and from the computational points of view.

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

We are currently working on finding other computational results, e.g., special cases of Theorem 1 which call for first-order, instead of second-order, equivalence. Moreover, 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., [12], we plan to extend our definitions in the traditional database context.

6 Acknowledgements

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

References

1. J.F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.
2. Egon Börger, Erich Gräedel, and Yuri Gurevich. *The Classical Decision Problem.* Perspectives in Mathematical Logic. Springer, 1997.
3. Marco Cadoli and Toni Mancini. Using a theorem prover for reasoning on constraint problems. In *Proceedings of the Ninth Conference of the Italian Association for Artificial Intelligence (AI*IA 2005)*, volume 3673 of *Lecture Notes in Artificial Intelligence*, pages 38–49. Springer, 2005.
4. H. Cambazard and N. Jussien. Integrating Benders decomposition within constraint programming. In *Proceedings of the Eleventh International Conference on Principles and Practice of Constraint Programming (CP 2005)*, number 3709 in Lecture Notes in Artificial Intelligence, pages 752–756. Springer-Verlag, 2005.
5. Pierluigi Crescenzi, Deborah Goldman, Christos H. Papadimitriou, Antonio Piccolboni, and Mihalis Yannakakis. On the complexity of protein folding. *J. of Comp. Biology*, 5(3):423–466, 1998.
6. Rina Dechter. Constraint Networks. In Stuart C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, volume 1. Addison-Wesley Publishing Company, 1992.
7. R. Fagin. Generalized First-Order Spectra and Polynomial-Time Recognizable Sets. In R. M. Karp, editor, *Complexity of Computation*, pages 43–74. AMS, 1974.
8. J. Hooker. *Logic-based methods for optimization: combining optimization and constraint satisfaction.*, chapter 19, pages 389–422. Wiley and Sons, 2000.
9. J.N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96:33–60, 2003.
10. V. Jain and I.E. Grossmann. Algorithms for hybrid MILP/CP models for a class of optimization problems. *INFORMS Journal on Computing*, 13:258–276, 2001.
11. Kit Fun Lau and Ken A. Dill. A lattice statistical mechanics model of the conformational and sequence spaces of proteins. *Macromolecules*, 22:3986–3997, 1989.
12. P. S. Medcraft, U. Schiel, and C.S. Baptista. Database integration using mobile agents. Number 2872 in Lecture Notes in Artificial Intelligence, pages 160–167. Springer-Verlag, 2003.
13. C. H. Papadimitriou. *Computational Complexity.* Addison Wesley, Reading, MA, 1994.
14. J.F. Puget. A fast algorithm for the bound consistency of alldiff constraints. In *AAAI/IAAI*, pages 359–366, 1998.
15. Pascal Van Hentenryck. *The OPL Optimization Programming Language.* The MIT Press, 1999.
16. T. Walsh. Permutation problems and channelling constraints. In *Proceedings of the Eighth International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2001)*, number 2250 in Lecture Notes in Computer Science, pages 377–391. Springer, 2001.