Journal of Heuristics, 8: 25–41, 2002

Soft Constraint Logic Programming and Generalized Shortest Path Problems*

STEFANO BISTARELLI AND UGO MONTANARI
Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56125 Pisa, Italy
email: bista@di.unipi.it
email: ugo@di.unipi.it

FRANCESCA ROSSI
Dipartimento di Matematica Pura ed Applicata, Università di Padova, Via Belzoni 7, 35131 Padova, Italy
email: frossi@math.unipd.it

Abstract

In this paper we study the relationship between Constraint Programming (CP) and Shortest Path (SP) problems. In particular, we show that classical, multicriteria, partially ordered, and modality-based SP problems can be naturally modeled and solved within the Soft Constraint Logic Programming (SCLP) framework, where logic programming is coupled with soft constraints. In this way we provide this large class of SP problems with a high-level and declarative linguistic support whose semantics takes care of both finding the cost of the shortest path(s) and also of actually finding the path(s). On the other hand, some efficient algorithms for certain classes of SP problems can be exploited to provide some classes of SCLP programs with an efficient way to compute their semantics.

Key Words: shortest path problems, soft constraints, constraint logic programming

1. Introduction

Shortest Path (SP) problems (Dreyfus, 1969; Leiserson, Cormen and Rivest, 1990) are among the most studied network optimization problems. They are mainly used to represent and solve transportation problems, where the optimization may involve different criteria, says cost, time, resources, etc. Most interesting is the multi-criteria case, where the optimization involves a set of criteria to be all optimized (Pallottino and Scutellà, 1988).

In this paper, we propose the Soft Constraint Logic Programming (SCLP) framework (Bistarelli, Montanari and Rossi, 1997; Georget and Codognet, 1998) as a linguistic support and a high-level and flexible programming environment where to model SP problems naturally and solve them efficiently.

SCLP programs are logic programs (Lloyd, 1987) where each ground atom can be seen as an instantiated soft constraint (Bistarelli, Montanari and Rossi, 1995, 1997b) and it can be associated with an element taken from a set. Formally, this set is a semiring, that is,

*Research partially supported by Italian MURST projects "Tecniche Formali per Sistemi Software" and "Concurrency, Higher-Order and Types".

a set plus two operations, $+$ and $\times$, which basically say how to combine constraints and how to compare them. The presence of these two operations allows to replace the usual boolean algebra for logic programming with a more general algebra where *logical and* and *logical or* are replaced by the two semiring operations. In this way, the underlying logic programming engine provides a natural tool to specify and solve combinatorial problems (like SP problems), while the soft constraint machinery provides greater expressivity and flexibility. In particular, the flexibility comes from the fact that the same SCLP programming environment, and operational semantic engine, can be used with different semirings (thus using different kinds of soft constraints), as we will see later in this paper.

Here we consider several versions of SP problems, from the classical one to the multi-criteria case, from partially ordered SP problems to those that are based on modalities, and we show how to model and solve them via SCLP programs. The basic idea is that soft constraints allow to faithfully represent the optimization criteria, and CLP provides a declarative way to describe the given SP problem. Moreover, this way of modeling and solving SP problems allows to associate with such problems both a declarative and an operational semantics.

The main results of the paper are as follows:

- Both classical, multi-criteria, partially-ordered, and modality-based SP problems are given a modelization as SCLP programs; such programs are able to both find the cost of the shortest path(s) and also a shortest path itself.
- A general methodology is provided to find a non-dominated path for both multi-criteria and partially-ordered SP problems; this methodology is based on a change of semiring, but does not require any change in the underlying SCLP syntax and semantics.
- A new algorithm to obtain the semantics for a particular class of SCLP programs is given, which is obtained by using a generalized version of the Floyd-Warshall algorithm (Floyd, 1962) for SP problems.

The paper is organized as follows: Section 2 gives the basic notions of soft constraints and SCLP programs, Section 3 shows the construction to pass from a classical SP problem to a CLP program, while Section 4 considers multi-criteria and partially-ordered SP problems, and Section 5 deals with modality-based SP problems. Then, in Section 6 we provide a class of SCLP programs with an efficient algorithm to compute their semantics, and finally Section 7 concludes the paper by summarizing its contributions and hinting at interesting directions for future work.

2. Soft constraint logic programming

In this section we describe the *soft constraint logic programming* (SCLP) framework, which we will later use to to describe SP problems in a declarative fashion. All the formal details that are skipped here can be found in Bistarelli, Montanari and Rossi (1997a, 1997b).

The SCLP framework is based on the notion of *c-semiring* introduced in Bistarelli, Montanari and Rossi (1995, 1997b). A c-semiring S is a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ where A is a set with two special elements ($\mathbf{0}, \mathbf{1} \in A$) and with two operations $+$ and $\times$ that satisfy

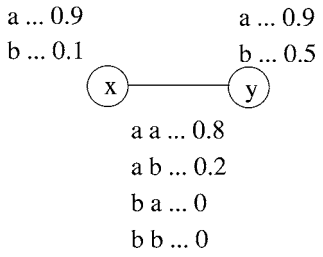

Figure 1. A fuzzy CSP.

certain properties: $+$ is closed, commutative, associative, idempotent, **0** is its unit element and **1** is its absorbing element; $\times$ is closed, associative, commutative, distributes over $+$, **1** is its unit element, and **0** is its absorbing element.

The $+$ operation defines a partial order $\leq_S$ over A such that $a \leq_S b$ iff $a + b = b$; when $a \leq_S b$ holds, we say that b is *better* than a. Other properties related to the two operations are: $+$ and $*$ are monotone on $\leq_S$; **0** is its minimum and **1** its maximum; $\langle A, \leq_S \rangle$ is a complete lattice and $+$ is its lub. Finally, if $\times$ is idempotent then: $+$ distribute over $\times$; $\langle A, \leq_S \rangle$ is a complete distributive lattice and $\times$ its glb.

Semiring-based constraint satisfaction problems (SCSPs) are constraint problems where each variable instantiation is associated with an element of a c-semiring A (to be interpreted as a cost, level of preference, ...), and constraints are combined via the $\times$ operation and compared via the $\leq_S$ ordering. Varying the set A and the meaning of the $+$ and $\times$ operations, we can represent many different kinds of problems, having features like fuzziness (using the semiring $S_{FCSP} = \langle \{x \mid x \in [0, 1]\}, max, min, 0, 1 \rangle$), probability ($S_{prob} = \langle \{x \mid x \in [0, 1]\}, max, \times, 0, 1 \rangle$), and optimization ($S_{WCSP} = \langle \mathcal{R}^+, \min, +, +\infty, 0 \rangle$). Moreover, the Cartesian product of two c-semirings is another c-semiring, and this can be fruitfully used to describe multi-criteria constraint satisfaction and optimization problems.

Figure 1 shows an example of a fuzzy SCSP. In this figure, variables are circles, constraints are arcs, and semiring values are written to the right of the corresponding tuples. The solutions of this SCSP are pairs of values (that is, a value for each of the two variables) and, according to the c-semiring $S_{FCSP} = \langle \{x \mid x \in [0, 1]\}, \max, \min, 0, 1 \rangle$, their semiring value is obtained by looking at the smallest value for all the subtuples (as many as the constraints) forming the pair.

The SCLP framework (Bistarelli, Montanari and Rossi, 1997a) extends the classical constraint logic programming formalism (Jaffar and Lassez, 1987) in order to handle also SCSP (Bistarelli, Montanari and Rossi, 1995, 1997b) problems. In passing from CLP to SCLP languages, we replaced classical constraints with the more general SCSP constraints where we are able to assign a *level of preference* to each instantiated constraint (i.e. a ground atom). To do this, we also modified the notions of interpretation, model, model intersection, and others, since we had to take into account the semiring operations and not the usual CLP operations. For example, while CLP interpretations associate a truth value (either *true* or *false*) with each ground atom, in SCLP ground atoms must be given one of the elements of the semiring. Also, while in CLP the value associated with an existentially quantified atom is the *logical or* among the truth values

Table 1. A simple example of an SCLP program.

```
s(X)    :- p(X,Y).
p(a,b)  :- q(a).
p(a,c)  :- r(a).
q(a)    :- t(a).
t(a)    :- 2.
r(a)    :- 3.
```

associated with each of its instantiations, here we had to replace the *or* with another operation which refers to one of the semiring operations (the $+$). The combination of atoms, which in CLP is modeled via *logical and*, in SCLP is instead handled via the $\times$ operation of the semiring.

Besides the model-theoretic semantics based on models and interpretations, SCLP programs come also with a fixpoint and an operational semantics. These semantics are conservative extensions of the corresponding ones for logic programming (LP), since by choosing a particular semiring (the one with just two elements, *true* and *false*, and the logical *and* and *or* as the two semiring operations) we get exactly the LP semantics.

The three semantics have been shown to be equivalent. In particular, we have proven that, given the set of all refutations starting from a given goal, it is possible to derive the declarative meaning of both the existential closure of the goal and its universal closure.

A simple example of an SCLP program over the semiring $\langle N, min, +, +\infty, 0\rangle$, where N is the set of non-negative integers and $D = \{a, b, c\}$, is represented in Table 1. The choice of this semiring allows to represent constraint optimization problems where the semiring elements are the costs for the instantiated atoms. Note that the ordering $\leq_S$ in this semiring coincides with the $\geq$ ordering over integers. The intuitive meaning of a semiring value like 3 associated with the atom $r(a)$ is that $r(a)$ costs 3 units. Thus the set N contains all possible costs, and the choice of the two operations min and $+$ implies that we intend to minimize the sum of the costs. This gives us the possibility to select the atom instantiation which gives the minimum cost overall.

Given a goal like $s(x)$ to this program, the semantics collects both a substitution for x (in this case, $x = a$) and also a semiring value (in this case, 2) which represents the minimum cost among the costs for all refutations for $s(x)$. Figure 2 shows the tree structure representing the two possible refutations for the goal $s(x)$. Each node in the tree represents a state of the computation, and the root represents the initial state. Each computation state contains both the current goal and a semiring value. At the beginning, the semiring value is the best one (0 in this example). Then, at each step, one clause or fact is used, and this changes the current goal, accumulates a substitution (written on the links of the tree), and combines the current semiring value with a new value given by the used clause. If a clause has no semiring value (like the first four clauses of the example), the associated value is assumed to be the best one. The combination of these two semiring values is achieved via the multiplicative operation of the semiring, which in this example is the sum. All the semiring values obtained by the refutations of the same goal (and with the same substitution) are then combined together via the additive operation of the semiring, which in this example

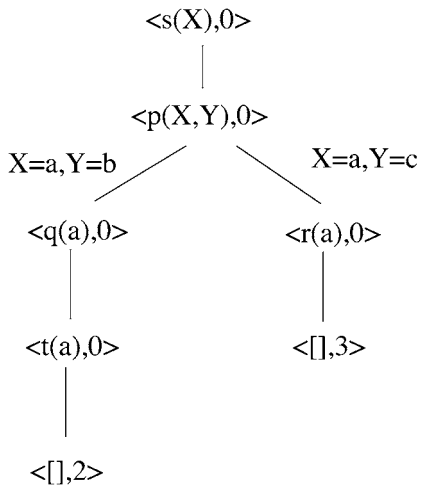

Figure 2. The SCLP refutation tree for goal $s(X)$.

is the min operation. Therefore the two refutations in Figure 2 produce the semiring value $\min(2, 3) = 2$. The formal definition of the operational semantics of SCLP programs can be found in Bistarelli, Montanari and Rossi (1997a).

The presence of a possibly partial order among the elements of the semiring makes the operational semantics more complex to compute. In fact, there could be several refutations for a goal which lead to different semiring elements which are not comparable in the partial order. In this case, these elements have to be combined in order to get the solution corresponding to the given goal, and their combination could be not reachable by any refutation path in the search tree.

The fact that we have to combine several refutation paths when we have a partial order (instead of a total one) can be fruitfully used in the context of this paper when we have an SP problem with incomparable costs in the arcs. In fact, in the case of a partial order, the solution of an SP problem should consists of all those paths which are not "dominated" by others.

SCLP programming has been implemented. Its implementation has been described in Georget and Codognet (1998), and the system `clp(FD,S)` is available for free. More information about the `clp(FD,S)` system, including also the instructions on how to down-load it, can be found at the URL `http://pauillac.inria.fr/~georget/clpfds.html`.

3. Classical SP problems

A shortest path (SP) problem can be represented as a directed graph $G = (N, E)$, where each arc $e \in E$ from node p to node q $(p, q \in N)$ has associated a label representing the cost of the arc from p to q.

There are four versions of the problem: the single pair problem, the single source problem, the single sink problem, and the all pair problem. However, the single source and the single sink problems are directional duals of each other, the single pair problem at least partially solves the single source/sink problem, and one way to solve the all pair problem is to solve

n single source/sink problems. For these reasons, the single sink (source) is fundamental and we concentrate on it. Given a set S of nodes and any node v in S (the sink), a solution of the problem consists of finding a path (or a set of paths) between any node of S and v, whose cost is minimal.

Consider for example the SP problem represented in figure 3: each arc has associated a label representing the cost (in money, time, space, ...) of that arc. In this example node v is the sink. Thus, given any node, we want to find a path from this node to the sink v (if it exists) that minimizes the cost.

To represent the classical version of SP problems, we consider SCLP programs over the semiring $S = \langle N, \min, +, +\infty, 0\rangle$, which, as noted above, is an appropriated framework to represent constraint problems where one wants to minimize the sum of the costs of the solutions.

From any SP problem we can build an SCLP program as follows. For each arc we have two clauses: one describes the arc and the other one its cost. More precisely, the head of the first clause represents the starting node, and its body contains both the final node and a predicate, say c, representing the cost of the arc. Then, the second clause is a fact associating to predicate c its cost (which is a semiring element). For example, if we consider the arc from p to q with cost 2, we have the clause

`p :-` c_{pq} `, q.`

and the fact

c_{pq} `:- 2.`

Finally, we must code that we want v to be the final node of the path. This is done by adding a clause of the form `v :- 0.` Note also that any node can be required to be the final one, not just those nodes without outgoing arcs (like v is in this example). The whole program corresponding to the SP problem in figure 3 can be seen in Table 2.

To compute a solution of the SP problem it is enough to perform a query in the SCLP framework; for example, if we want to compute the cost of the path from r to v we have to perform the query `:-r`. For this query, we obtain the value 6, that represents the cost of the best path(s) from r to v.

Notice that to represent classical SP problems in SCLP, we do not need any variable. Thus the resulting program is propositional. However, this program, while giving us the

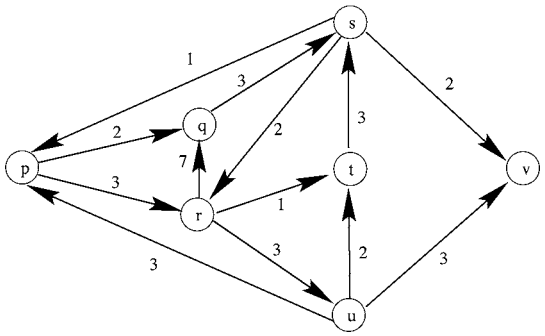

Figure 3. An SP problem.

Table 2. The SCLP program representing the SP problem in figure 3.

p :- c_{pq}, q.	c_{pq} :- 2.
p :- c_{pr}, r.	c_{pr} :- 3.
q :- c_{qs}, s.	c_{qs} :- 3.
r :- c_{rq}, q.	c_{rq} :- 7.
r :- c_{rt}, t.	c_{rt} :- 1.
r :- c_{ru}, u.	c_{ru} :- 3.
s :- c_{sp}, p.	c_{sp} :- 1.
s :- c_{sr}, r.	c_{sr} :- 2.
s :- c_{sv}, v.	c_{sv} :- 2.
t :- c_{ts}, s.	c_{ts} :- 3.
u :- c_{up}, p.	c_{up} :- 3.
u :- c_{ut}, t.	c_{ut} :- 2.
u :- c_{uv}, v.	c_{uv} :- 3.
v :- 0.	

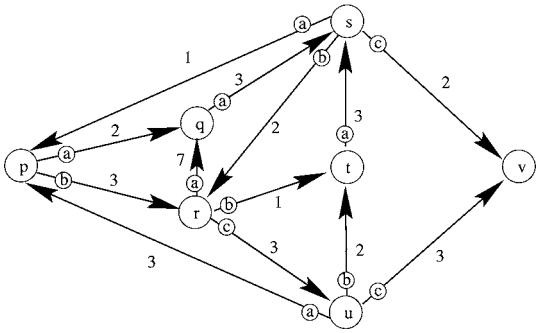

Figure 4. An SP problem with labeled arcs.

cost of the shortest paths, does not give us any information about the arcs which form such paths. This information could be obtained by providing each predicate with an argument, which represents the arc chosen at each step.

Figure 4 shows the same SP problem of figure 3 where the arcs outgoing each node have been labeled with different labels to distinguish them. Such labels can then be coded into the corresponding SCLP program to "remember" the arcs traversed during the path corresponding to a solution. For example, clause

p :- c_{pq}, q.

would be rewritten as

p(a) :- c_{pq}, q(X).

Here constant a represents one of the arcs going out of p: the one which goes to q. If all clauses are rewritten similarly, then the answer to a goal like :-r(X) will be both a

semiring value (in our case 6) and a substitution for X. This substitution will identify the first arc of a shortest path from r to v. For example, if we have $X = b$, it means that the first arc is the one that goes from r to t. To find a complete shortest path, we just need to compare the semiring values associated with each instantiated goal, starting from r and following the path. For example, in our case (of the goal $\exists X.r(X)$) we have that the answer to the goal will be $X = c$ with semiring value 6. Thus we know that a shortest path from r to v can start with the arc from r to u. To find the following arc of this path, we compare the semiring values of $u(a)$, $u(b)$, and $u(c)$. The result is that $u(c)$ has the smallest value, which is 3. Thus the second arc of the shortest path we are constructing is the one from u to v. The path is now finished because we reached v which is our sink.

Notice that a shortest path could be found even if variables are not allowed in the program, but more work is needed. In fact, instead of comparing different instantiations of a predicate, we need to compare the values associated with the predicates that represent nodes reachable by alternative arcs starting from a certain node, and sum them to the cost of such arcs. For example, instead of comparing the values of $p(a)$ and $p(b)$ (figure 4), we have to compare the values of $q + 2$ and of $r + 3$ (figure 3).

A third alternative to compute a shortest path, and not only its cost, is to use lists: by replacing each clause of the form

```
p :- c_xy, q.
```

with the clause

```
p([a|T]):-c_xy, q(T).
```

during the computation we also build the list containing all arcs which constitute the corresponding path. Thus, by giving the goal `:-p(L).`, we would get both the cost of a shortest path and also the shortest path itself, represented by the list L.

An alternative representation, probably more familiar for CLP-ers, of SP problems in SCLP is one where there are facts of the form

```
c(p,q):-2.
⋮
c(u,v):-3.
```

to model the graph, and the two clauses

```
path(X,Y):-c(X,Y).
path(X,Y):-c(X,Z), path(Z,Y).
```

to model paths of length one or more. In this representation the goal to be given to find the cost of the shortest path from p to v is `:-path(p,v)`. This representation is obviously more compact than the one in Table 2, and has equivalent results and properties. However, in this paper we will use with the simpler representation, used in Table 2, where all clauses have at most one predicate in the body. The possibility of representing SP problems with SCLP programs containing only such a kind of clauses is important, since it will allow us to use efficient algorithms to compute the semantics of such programs (see Section 6 for more details).

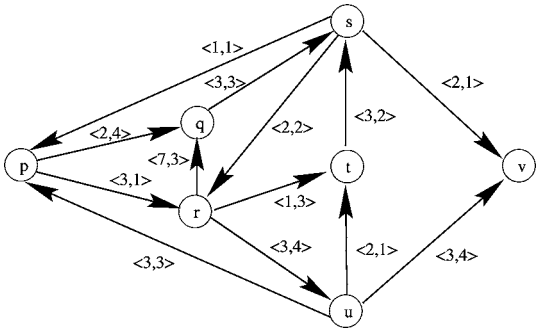

Figure 5. A multi-criteria SP problem.

4. Partially-ordered SP problems

Sometimes, the costs of the arcs are not elements of a totally ordered set. A typical example is obtained when we consider multi-criteria SP problems.

For example, if an arc represents a piece of highway between two cities, then we can label each arc both with the cost and with the time needed to follow this piece. Then, we want to find the paths which have both the minimum overall cost and the minimum overall time. In this example, there may be cases in which the labels of two arcs are not compatible, like $\langle \$5, 20' \rangle$ and $\langle \$7, 15' \rangle$. In general, when we have a partially ordered set of costs, it may be possible to have several paths, all of which are not *dominated* by others, but which have different incomparable costs.

Consider for example the multi-criteria SP problem shown in figure 5: each arc has associated a pair representing the weight of the arc in terms of *cost* and *time*. Given any node, we want to find a path from this node to v (if it exists) that minimizes both criteria.

We can translate this SP problem into the SCLP program in Table 3. This program works over the semiring

$$\langle N^2, \text{min'}, +', \langle +\infty, +\infty \rangle, \langle 0, 0 \rangle \rangle,$$

where min' and $+'$ are classical min and $+$, suitably extended to pairs. In practice, this semiring is obtained by putting together, via the Cartesian product, two instances of the semiring $\langle N, \text{min}, +, +\infty, 0 \rangle$ (we recall that the Cartesian product of two c-semirings is a c-semiring as well (Bistarelli, Montanari and Rossi, 1997b)). On of the instances is used to deal with the cost criteria, the other one is for the time criteria. By working on the combined semiring, we can deal with both criteria simultaneously: the partial order will tell us when a pair $\langle$cost, time$\rangle$ is preferable to another one, and also when they are not comparable.

To give an idea of another practical application of partially ordered SP problems, just think of network routing problems where we need to optimize according to the following criteria: minimize the time, minimize the cost, minimize the number of arcs traversed, and maximize the bandwidth. The first three criteria correspond to the same semiring, which is $\langle N, min, +, +\infty, 0 \rangle$, while the fourth criteria can be characterized by the semiring

Table 3. The SCLP program representing the multi-criteria SP problem in figure 5.

p :- c_{pq}, q.	c_{pq} :- < 2,4 >.
p :- c_{pr}, r.	c_{pr} :- < 3,1 >.
q :- c_{qs}, s.	c_{qs} :- < 3,3 >.
r :- c_{rq}, q.	c_{rq} :- < 7,3 >.
r :- c_{rt}, t.	c_{rt} :- < 1,3 >.
r :- c_{ru}, u.	c_{ru} :- < 3,4 >.
s :- c_{sp}, p.	c_{sp} :- < 1,1 >.
s :- c_{sr}, r.	c_{sr} :- < 2,2 >.
s :- c_{sv}, v.	c_{sv} :- < 2,1 >.
t :- c_{ts}, s.	c_{ts} :- < 3,2 >.
u :- c_{up}, p.	c_{up} :- < 3,3 >.
u :- c_{ut}, t.	c_{ut} :- < 2,1 >.
u :- c_{uv}, v.	c_{uv} :- < 3,4 >.
v :- < 0,0 >.	

$\langle [0, U], max, min, 0, U\rangle$, where U is the maximal bandwidth in an arc. In this example, we have to work on a semiring which is obtained by vectorizing all these four semirings. Each of the semirings is totally ordered but the resulting semiring, whose elements are four-tuples, is partially ordered.

Notice that the only difference between the structure of the program in Table 3 and the one in the previous section is that here we have costs represented by pairs of values, and, since we have a partial order, two such pairs may possibly be incomparable. This may lead to a strange situation while computing the semantics of a given goal. For example, if we want to compute the cost and time of a best path from p to v, by giving the query `:-p.`, the resulting answer in this case is the value $\langle 7, 7\rangle$. While the semiring value obtained in totally ordered SCLP programs represented the cost of one of the shortest paths, here it is possible that there are no paths with this cost: the obtained semiring value is in fact in general the greatest lower bound of the costs of all the paths from p to v. This behavior comes from the fact that, if different refutations for the same goal have different semiring values, the SCLP framework combines them via the $+$ operator of the semiring (which, in the case of our example, is the min' operator). If the semiring is partially ordered, it may be that $a + b$ is different from both a and b. On the contrary, if we have a total order $a + b$ is always either a or b.

This of course is not satisfactory, because usually one does not want to find the greatest lower bound of the costs of all paths from the given node to the sink, but rather prefers to have one of the non-dominated paths. To solve this problem, we can add variables to the SCLP program, as we did in the previous section, and also change the semiring. In fact, we now need a semiring which allows us to associate with each node the set of the costs of all non-dominated path from there to the sink. In other words, starting from the semiring $S = \langle A, +, \times, 0, 1\rangle$ (which, we recall, in our case is $\langle N^2, min', +', \langle +\infty, +\infty\rangle, \langle 0, 0\rangle\rangle$), we now have to work with the semiring $P^H(S) = \langle P^H(A), \uplus, \times^*, \emptyset, A\rangle$, where:

- $P^H(A)$ is the Hoare Power Domain (Smyth, 1978) of A, that is, $P^H(A) = \{S \subseteq A \mid x \in S, y \leq_S x \text{ implies } y \in S\}$. In words, $P^H(A)$ is the set of all subsets of A which are downward closed under the ordering $\leq_S$. It is easy to show that such sets are isomorphic to those containing just the non-dominated values. Thus in the following we will use this more compact representation for efficiency purposes. In this compact representation, each element of $P^H(A)$ will represent the costs of all non-dominated paths from a node to the sink;
- the top element of the semiring is the set A (its compact form is $\{1\}$, which in our example is $\{\langle 0, 0\rangle\}$);
- the bottom element is the empty set;
- the additive operation $\uplus$ is the *formal union* (Smyth, 1978) that takes two sets and obtains their union;
- the multiplicative operation $\times^*$ takes two sets and produces another set obtained by multiplying (using the multiplicative operation $\times$ of the original semiring, in our case $+'$) each element of the first set with each element of the second one;
- the partial order of this semiring is as follows: $a \leq_{P^H(S)} b$ iff $a \uplus b = b$, that is for each element of a, there is an element in b which dominates it (in the partial order $\leq_S$ of the original semiring).

From the theoretical results in Smyth (1978), adapted to consider c-semirings, we can prove that $P^H(S)$ and its more compact form are indeed isomorphic. Moreover, we can also prove that given a c-semiring S, the structure $P^H(S)$ is a c-semiring as well.

Theorem 1. *[$P^H(S)$ is a c-semiring] Given a c-semiring $S = \langle A, +, \times, 0, 1\rangle$, the structure $P^H(S) = \langle P^H(A), \uplus, \times*, \emptyset, A\rangle$ obtained using the* Power domain of Hoare *operator is a c-semiring.*

Proof: It easily follows from the properties of the $\times$ operator in the c-semiring S and from the properties (commutativity, associativity, and idempotence) of the formal union $\uplus$ in $P^H(S)$. □

Note that in this theorem we do not need any assumption over the c-semiring S. Thus the construction of $P^H(S)$ can be done for any c-semiring S. Notice also that, if S is totally ordered, the c-semiring $P^H(S)$ does not give any additional information w.r.t. S. In fact, if we consider together the empty set (with the meaning that there are no paths) and the set containing only the bottom of A (with the meaning that there exists a path whose cost is 0), it is possible to build an isomorphism between S and $P^H(S)$ by mapping each element p (a set) of $P^H(A)$ onto the element a of A such that $a \in p$ and a dominates all elements in the set p.

The only change we need to make to the program with variables, in order to work with this new semiring, is that costs now have to be represented as singleton sets. For example, clause c_{pq} `:-<2, 4>.` will become c_{pq} `:-{<2, 4>}.`

Let us now see what happens in our example if we move to this new semiring. First we give a goal like `:-p(X)..` As the answer, we get a set of pairs, representing the costs of all non-dominated paths from p to v. All these costs are non-comparable in the partial

order, thus the user is requested to make a choice. However, this choice could identify a single cost or also a set of them. In this second case, it means that the user does not want to commit to a single path from the beginning and rather prefers to maintain some alternatives. The choice of one cost of a specific non-dominated path will thus be delayed until later. If instead the user wants to commit to one specific cost at the beginning, say $\langle c_1, c_2\rangle$, he/she then proceeds to find a path which costs exactly $\langle c_1, c_2\rangle$. By comparing the answers for all goals of the form $p(a)$, where a represents one of the arcs out of p, we can see which arc can start a path with cost $\langle c_1, c_2\rangle$. In fact, such an arc will be labeled $\langle c_{1a}, c_{2a}\rangle$ and will lead to a node with an associated set of costs $\langle c_3, c_4\rangle$ such that $\langle c_3, c_4\rangle \times \langle c_{1a}, c_{2a}\rangle = \langle c_1, c_2\rangle$. Suppose it is the arc from p to q, which has cost $\langle 7, 3\rangle$. Now we do the same with goals of the form $q(a)$, to see which is the next arc to follow. However, we now have to look for the presence of a pair $\langle c_3, c_4\rangle$ such that $\langle c_3, c_4\rangle \times \langle 7, 3\rangle = \langle c_1, c_2\rangle$.

Notice that each time we look for the next arc, we choose just one alternative and we disregard the others. If we used a fixpoint (or any bottom-up) semantics to compute the answer of the initial goal :- p(X)., then all the other answers we need for this method have been already computed, thus the method does not require any additional computational effort to find a non-dominated path.

Notice also that the sets of costs associated with each node are non-dominated. Thus the size of these sets in the worst case is the size of the maximal "horizontal slice" of the partial order: if we can have at most N non-dominated element in the partial order, then each of such sets will have size at most N. Of course in the worst case N could be the size of the whole semiring (when the partial order is completely "flat").

Most classical methods to handle multi-criteria SP problems find the shortest paths by considering each criteria separately, while our method deals with all criteria at once. This allows to obtain optimal solutions which are not generated by looking at each single criteria. In fact, some optimal solutions could be non-optimal in each of the single criteria, but still are incomparable in the overall ordering. Thus we offer the user a greater set of non-comparable optimal solutions. For example, by using the time-cost multi-criteria scenario, the optimal solution w.r.t. time could be 1 minute (with cost of 10 dollars), while the optimal solution w.r.t. cost could be 1 dollar (with time of 10 minutes). By considering both criteria together, we could also obtain the solution with 2 minutes and 2 dollars!

Finally, this method is applicable not only to the multi-criteria case, but to any partial order, giving us a general way to find a non-dominated path in a partially-ordered SP problem. It is important to notice here the flexibility of the semiring approach, which allows us to use the same syntax and computational engine, but on a different semiring, to compute different objects.

5. Modality-based SP problems

Until now we have considered situations in which an arc is labeled by its cost, be it one element or a tuple of elements as in the multi-criteria case. However, sometimes it may be useful to associate with each arc also information about the *modality* to be used to traverse the arc.

For example, interpreting arcs as links between cities, we may want to model the fact that we can cover such an arc by *car*, or by *train*, or by *plane*. Another example of a modality

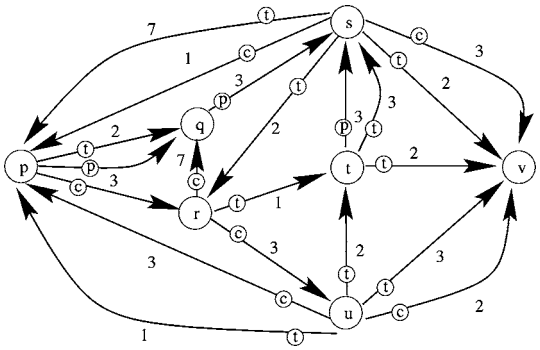

Figure 6. An SP problem with modalities.

could be the *time of the day* in which we cover the arc, like *morning*, *afternoon*, and *night*. In all these cases, the cost of an arc may depend on its modality.

An important thing to notice is that a path could be made of arcs which not necessarily are all covered with the same modality. For example, the path between two cities can be made of arcs, some of which are covered in the morning and others in the afternoon. Moreover, it can be that different arcs have different sets of modalities. For example, from city A to city B we can use both the car or the train, and from city B to city C we can use only the plane. Thus modalities cannot be simply treated by selecting a subset of arcs (all those with the same modality).

An example of an SP problem with the three modalities representing car (c), plane (p), and train (t) can be seen in figure 6. Here the problem is to find a shortest path from any node to v, and to know both its cost and also the modalities of its arcs.

This SP problem with modalities can be modeled via the SCLP program in Table 4. In this program, the variables represent the modalities.

If we ask the query `:-p(c).`, it means that we want to know the smallest cost for a trip from p to v using the *car*. The result of this query in our example is $p(c) = 9$ (using the path $p - r - u - v$).

Notice that the formulation shown in figure 4 puts some possibly undesired constraints on the shortest path to be found. In fact, by using the same variable in all the predicates of a rule, we make sure that the same modality (in our case the same transport mean) is used throughout the whole path. If instead we want to allow different modalities in different arcs of the path, then we just need to change the rules by putting a new variable on the last predicate of each rule. For example, rule

$$\texttt{p(X) :-} c_{pq}\texttt{(X), q(X).}$$

would become

$$\texttt{p(X) :-} c_{pq}\texttt{(X), q(Y).}$$

Now we can use a modality for the arc from p to q, and another one for the next arc. In this new program, asking the query `:-p(c).` means that we want to know the smallest cost for a trip from p to v using the car in the first arc.

Table 4. The SCLP program representing the SP problem with modalities.

p(X) :- c_{pq}(X), q(X).	c_{pq}(t) :- 2.
p(X) :- c_{pr}(X), r(X).	c_{pq}(p) :- 3.
q(X) :- c_{qs}(X), s(X).	c_{pr}(c) :- 3.
r(X) :- c_{rq}(X), q(X).	c_{qs}(p) :- 3.
r(X) :- c_{rt}(X), t(X).	c_{rq}(c) :- 7.
r(X) :- c_{ru}(X), u(X).	c_{rt}(t) :- 1.
s(X) :- c_{sp}(X), p(X).	c_{ru}(c) :- 3.
s(X) :- c_{sr}(X), r(X).	c_{sp}(c) :- 1.
s(X) :- c_{sv}(X), v(X).	c_{sp}(t) :- 7.
t(X) :- c_{ts}(X), s(X).	c_{sr}(t) :- 2.
u(X) :- c_{up}(X), p(X).	c_{sv}(t) :- 2.
u(X) :- c_{ut}(X), t(X).	c_{sv}(c) :- 3.
u(X) :- c_{uv}(X), v(X).	c_{ts}(p) :- 3.
v(X) :- 0.	c_{ts}(t) :- 3.
	c_{up}(c) :- 3.
	c_{up}(t) :- 1.
	c_{ut}(t) :- 2.
	c_{uv}(t) :- 3.
	c_{uv}(c) :- 2.

The same methods used in the previous sections to find a shortest path, or a non-dominated path in the case of a partial order, can be used in this kind of SCLP programs as well. Thus we can put additional variables in the predicates to represents alternative arcs outgoing the corresponding nodes, and we can shift to the semiring containing sets of costs to find a non-dominated path. In particular, a clause like

p(X):-c_{pq}(X), q(Y).

would be rewritten as

p(X,a):-c_{pq}(X), q(Y,Z).

6. An SP algorithm for a class of SCLP programs

Summarizing what we did in the previous sections, we can say that the general form of the SCLP programs we use to represent SP problems consists of several clauses for each predicate p_i, where each clause body has one constraint and one other predicate p_j, plus one special clause for the sink predicate (with a **0** in the body), plus the facts defining the costs of the arcs. Table 5 shows this general form (only the clauses and without variables). It is important to notice that these are not general SCLP programs, since there is always one predicate symbol in each clause body (since the constraint can be replaced by its cost). In the logic programming literature, these programs are called *linear*, because of the restriction on the number of predicates in the bodies of the clauses.

Table 5. The general form of an SCLP program representing an SP problem.

p_1 :- c_{12}, p_2.
⋮
p_1 :- c_{1n}, p_n.
p_2 :- c_{12}, p_1.
p_2 :- c_{13}, p_3.
⋮
p_2 :- c_{2n}, p_n.
⋮
p_v :- **0**.
⋮
p_n :- c_{n1}, p_1.
⋮
p_n :- $c_{n,n-1}$, p_{n-1}.

Table 6. The system of equations corresponding to the SCLP program form of Table 5.

$p_1 =$		$c_{12} \times p_2 + \ldots$	$+ c_{1n} \times p_n$	
$p_2 = c_{21} \times p_1 +$		$c_{23} \times p_3 + \ldots$	$+ c_{2n} \times p_n$	
⋮				
$p_v =$				**0**
⋮				
$p_n = c_{n1} \times p_n + \ldots$			$+ c_{nn-1} \times p_{n-1}$	

Given any SCLP program of the form shown in Table 5, its semantics can be obtained using classical methodologies (for the CLP literature) as described in Section 2. However, we will now show an additional method, based on a classical algorithm for SP problems, suitable adapted to c-semirings, to obtain the same semantics. The algorithm we will use is the one developed by Floyd and Warshall (1962). To put the SCLP program into a shape which can be handled by this algorithm, we first need to transform it into a system of linear equations, one for each predicate: the left hand side of the equation is the chosen predicate, while the right hand side is obtained by combining the bodies of all the clauses defining the predicates via the $+$ operator, and replacing the comma with the $\times$ operator in each body. The result of this transformation, applied to the program of Table 5, can be seen in Table 6.

From this system of equations we can now build a graph with a node i for each predicate p_i, and where the cost of the arc between nodes i and j is given by predicate c_{ij}. In this

way, this system of equations can be interpreted also as the matrix showing, for each pair of nodes, the cost of the arc (if it exists) between such nodes. In fact, each column (and row) can be associated with one node.

Given this matrix, the algorithm works by selecting each triple of nodes, say i, j and k, and performing the following assignment: $c_{ij} := c_{ij} + (c_{ik} \times c_{kj})$, where the $+$ and $\times$ operators in this assignment refer to the operations of the chosen semiring.

The triples are chosen by first selecting a value for index k (which is the intermediate node in the two-step path), and then varying the indices i and j in all possible ways (to consider all two-step paths from i to j). After considering all values for k, that is, after n^3 steps, the value of the element in row i and column v represents the cost of the shortest path from node i to v. In terms of SCLP semantics, this is the semantics of predicate p_i.

This method to obtain the semantics of an SCLP program of the form in Table 5 can be used independently of the semiring underlying the given program. In fact, as noted above, the version we use of the original Floyd-Warshall algorithm does not depend on the meaning of the $+$ and $\times$ operators. In particular, it can also be used for partially ordered semirings. However, if the semiring we consider is totally ordered, we can also use any other classical algorithm for solving SP problems.

Summarizing, this section provides a class of SCLP programs, over any semiring, with a new method to obtain the semantics whose complexity is cubic in the number of predicates of the program.

The form of the SCLP program in Table 5 does not contain any variable. However, our method works also when each predicate (both the p_i and the c_{ij} predicates) has some variables, provided that, in each clause, all the predicates have the same variables in the same order. Thus, for example, we can admit clauses of the form $p(X, Y, Z) : - c(X, Y, Z)$, $q(X, Y, Z)$. This kind of clauses can be handled by passing to a more complex semiring, where each element is a vector (or a matrix) of the elements of the original semiring, and the two operations are vectorizations of the original ones.

7. Conclusions

We have investigated the relationship between shortest path problems and constraint programming, proposing the soft constraint logic programming framework to model and solve many versions of this problem in a declarative way. In particular, both classical, multi-criteria, partially-ordered, and modality-based SP problems are shown to be expressible in this programming formalism.

We also have shown that certain algorithm to solve SP problems can be used to compute efficiently the semantics of a certain class of SCLP programs. Such a class could be extended by using structures more complex than semirings, or by considering trees instead of paths. We plan to investigate these structures in the future.

In particular, we plan to use SCLP programs also to model and solve best tree problems. We also intend to apply our methodology to specify and solve planning problems, since in many cases such problems can be cast as best tree problems over the graph describing all possible moves of the planning scenario.

References

Bistarelli, S., U. Montanari, and F. Rossi. (1995). "Constraint Solving over Semirings." In *proc. IJCAI95*. San Matco, CA: Morgan Kaufman.

Bistarelli, S., U. Montanari, and F. Rossi. (1997a). "Semiring-based Constraint Logic Programming." In *proc. IJCAI97*. San Matco, CA: Morgan Kaufman.

Bistarelli, S., U. Montanari, and F. Rossi. (1997b). "Semiring-based Constraint Solving and Optimization." *Journal of the ACM* 44(2), 201–236.

Dreyfus, S.E. (1969). "An Appraisal of Some Shortest-Paths Algorithms." *Operation Research* 17, 395–412.

Floyd, R.W. (1962). Algorithm 97: Shortest Path: *Communication of ACM* 5, 345.

Georget, Y. and P. Codognet. (1998). "Compiling Semiring-Based Constraints with clp (fd,s)." In *CP98*, Springer. LNCS 1520.

Jaffar, J. and J.L. Lassez. (1987). "Constraint Logic Programming." In *Proc. POPL*. New York: ACM.

Leiserson, C.E., T.H. Cormen, and R.L. Rivest. (1990). *Introduction to Algorithms*, ch. 26. Cambridge, MA: MIT Press.

Lloyd, J.W. (1987). *Foundations of Logic Programming*. Berlin: Springer Verlag.

Pallottino, S. and M.G. Scutellà. (1988). "Shortest Path Algorithms in Transportation Models: Classical and Innovative Aspects. In P. Marcotte and S. Nguyen (eds.), *Equilibrium and Advanced Transportation Modelling*. pp. 245–281.

Smyth, M.B. (1978). "Power Domains." *Journal of Computer and System Sciences* 16(1), 23–36.