A Parallel LP$^{\mathrm{MLN}}$ Solver: Primary Report

Bin Wang and Zhizheng Zhang

School of Computer Science and Engineering,
Southeast University, Nanjing 211189, China,
{kse.wang, seu_zzz}@seu.edu.cn

Abstract. LP$^{\mathrm{MLN}}$ that incorporates Answer Set Programming (ASP) and Markov Networks is a powerful tool for handling uncertain and non-monotonic knowledge. In this paper, we propose a parallelized method for solving LP$^{\mathrm{MLN}}$ programs. The main idea of the method is partitioning a ground LP$^{\mathrm{MLN}}$ program into several corresponding programs, called augmented subsets, where each augmented subset will be translated into an ASP program, hence stable models and their penalties of all these augmented subsets can be generated concurrently by calling efficient ASP solvers. Then, the LP$^{\mathrm{MLN}}$ solver computes the stable models and their weights of the LP$^{\mathrm{MLN}}$ program from the stable models and their penalties of all augmented subsets of the LP$^{\mathrm{MLN}}$ program, and gives the results of an inference task. We present the approaches to the partition and the translation, and give some theoretical results that guarantee the correctness of the parallelized method. Experimental results show that the parallelized method can improve performance of solving LP$^{\mathrm{MLN}}$ programs.

Keywords: parallel, LP$^{\mathrm{MLN}}$ solver, Answer Set Programming

1 Introduction

LP$^{\mathrm{MLN}}$ [12] is a newly introduced knowledge representation language that combines the ideas of Answer Set Programming (ASP) [7] and Markov Logic Networks (MLN) [18]. The researches in [10,12,13] show the powerful expressivity of LP$^{\mathrm{MLN}}$, which is able to embed MLN and several probabilistic logic languages, such as ProbLog [4], Pearls' Causal Models [17], and P-log [2].

So far as we know, several approaches to the implementation of the LP$^{\mathrm{MLN}}$ solver have been presented. The basic idea of all the approaches is to compile the LP$^{\mathrm{MLN}}$ language into another formalism that has efficient solvers. Lee and Wang [12] extended the concept of completion and tightness from ASP to LP$^{\mathrm{MLN}}$, which can be used to translate a tight LP$^{\mathrm{MLN}}$ program to an MLN program. Furthermore, through combining with the loop formulas technique [15,9], an arbitrary LP$^{\mathrm{MLN}}$ program can be translated into an MLN program such that MLN solvers such as Alchemy [8], Tuffy [16] etc. can be used for solving LP$^{\mathrm{MLN}}$ programs. Balai and Gelfond [1] presented a translation from a subset of LP$^{\mathrm{MLN}}$ to P-log [2] that is a probabilistic extension of ASP such that LP$^{\mathrm{MLN}}$ programs can be solved by using P-log solvers. Most recently, Lee and Yang [13] proposed a translation from LP$^{\mathrm{MLN}}$ to a set of weak constraints [3] and choice formulas that can be translated into an ASP program, such that Maximum A Posteriori probability (MAP) estimates of an LP$^{\mathrm{MLN}}$ program can be solved by calling ASP solvers such as CLASP [6], DLV [14] etc. The inference of LP$^{\mathrm{MLN}}$ programs, especially exact inference, needs to compute all stable models of the program and then to do statistic computation on those stable models, which is time consuming. To implement an efficient LP$^{\mathrm{MLN}}$ solver, this paper investigates the parallelized method for solving an LP$^{\mathrm{MLN}}$ program.

In this paper, we aim at presenting a parallelized method for solving LP$^{\mathrm{MLN}}$ programs. The main idea of the method is partitioning a ground LP$^{\mathrm{MLN}}$ program into several corresponding programs, called augmented subsets, where each augmented subset will be translated into an ASP program, hence stable models and their penalties of all these augmented subsets can be generated concurrently by calling efficient ASP solvers. Then, the LP$^{\mathrm{MLN}}$ solver computes the stable models and their weights of the LP$^{\mathrm{MLN}}$ program from the stable models and their penalties of all augmented subsets of the LP$^{\mathrm{MLN}}$ program, and gives the results of an inference task.

Algorithm 1: parallel LP$^{\mathrm{MLN}}$ Solver

```
  Input: $M$: an LP$^{\mathrm{MLN}}$ program, $n$: number of processors, $q$: query
  Output: $R$: inference results
1 begin
      // stage 1: partition $M$ into $n$ augmented subsets
2     $S$ =Partition($M, n$) ;
3     for $S_i \in S$ do // for each processor do ...
          // stage 2:translate $S_i$ to an ASP program $\Pi^i$
4         $\Pi^i$ =Translation($M, S_i$);
          // stage 3: generate stable models and their penalties
5         $Penalty^i, AS^i$ =ASPSolver($\Pi^i$);
6         weight of $AS^i$ is $WeightComputing(Penalty^i)$;
      // stage 4: synthesis stage
7     $SM = \bigcup_{i=1}^{n} AS^i$;
8     $R$ =AnswerQuery($SM, q$);
9     return $R$;
```

At a high abstract level, as shown in Algorithm 1, our parallel solver will work with four stages: Partition, Translation, Stable Model Generation, and Synthesis. In the partition stage, an LP$^{\mathrm{MLN}}$ program is partitioned into several augmented subsets such that inference tasks of the LP$^{\mathrm{MLN}}$ program can be addressed via solving these augmented subsets concurrently. In the translation stage, each augmented subset is translated into an ASP program, which is an extension of the translation introduced in [13]. In the stable model generation stage, ASP programs are solved by an ASP solver, which enumerates all stable models and computes their penalties. In the synthesis stage, the LP$^{\text{MLN}}$ solver collects all stable models with penalties and computes results of inference tasks.

For now, there are two kinds of inference tasks supported by our solver: MAP task and PI task. MAP task requires solver to compute the most probable stable models of the input LP$^{\text{MLN}}$ program. And PI task requires solver to compute the marginal probabilities of some propositions w.r.t. the input LP$^{\text{MLN}}$ program. From Algorithm 1, we can see that the key problems in this work are how to partition an input program in the function **Partition** and how to translate an augmented subset into an ASP program in the function **Translation** such that the MAP task and PI task can be done.

Our main contributions are as follows. Firstly, we present an approach to partition an LP$^{\text{MLN}}$ program into several augmented subsets, which contains some theoretical results and a practical algorithm. Secondly, we present a translation from an augmented subset into an ASP programs, which preserves all stable models of the augmented subset, and the weight degree of a stable model in the sense of LP$^{\text{MLN}}$ can be derived from the penalties of a stable model in the sense of ASP.

The rest of this paper is organized as follows. Section 2 reviews LP$^{\text{MLN}}$ and weak constraints. Section 3 presents our partition approach including the theoretic foundation of partition and the corresponding algorithm. Section 4 presents our translation approach. Section 5 shows experimental results and gives some discussion on the partition. Section 6 concludes the paper by summarizing the obtained results and future work.

2 Preliminaries

2.1 LP$^{\text{MLN}}$

An LP$^{\text{MLN}}$ program is a finite set of rules of the form $w : r$, where w is either a real number or the symbol α denoting the "infinite weight", and r is an ASP rule of the form

$$l_1 \ \vee \ ... \ \vee \ l_k \ \leftarrow \ l_{k+1}, ..., \ l_m, \ not \ l_{m+1}, ..., \ not \ l_n. \tag{1}$$

where ls are literals, $\vee$ is epistemic disjunction, and *not* is default negation. A rule $w : r$ is called soft if w is a real number, and hard if w is α. We use $\overline{M}$ to denote the set of unweighted rules of an LP$^{\text{MLN}}$ program M, i.e. $\overline{M} = \{r | w : r \in M\}$. A ground LP$^{\text{MLN}}$ rule $w' : r'$ is obtained from its corresponding non-ground rule $w : r$ by replacing every variable of $w : r$ with every ground term of a signature, and the weight of rule $w' : r'$ is the same as the weight of rule $w : r$.

A ground LP$^{\text{MLN}}$ rule $w : r$ is satisfied by a consistent set X of ground literals, denoted by $X \models w : r$, if $X \models r$ by the notion of satisfiability in ASP. An LP$^{\text{MLN}}$ program M is satisfied by X, denoted by $X \models M$, if X satisfies all rules in M. We use M_X to denote the reduct of an LP$^{\text{MLN}}$ program M with respect to X, i.e. $M_X = \{w : r \in M \mid X \models w : r\}$. X is a stable model of the program M if X is a stable model of $\overline{M_X}$. We use $SM(M)$ to denote the set of all stable models of an LP$^{\mathrm{MLN}}$ program M. For a stable model X of an LP$^{\mathrm{MLN}}$ program M, the weight degree $W(M,X)$ of X w.r.t. M is defined as

$$W(M,X) = exp\left(\sum_{w:r\in M_X} w\right) \tag{2}$$

and the probability degree $P_s(M,X)$ of X w.r.t. M is defined as

$$P_s(M,X) = \lim_{\alpha\to\infty} \frac{W(M,X)}{\Sigma_{X'\in SM(M)} W(M,X')} \tag{3}$$

For a proposition β, its probability degree $P_p(M,\beta)$ w.r.t. M is defined as

$$P_p(M,\beta) = \sum_{X\in SM(M) \text{ and } X\models\beta} P_s(M,X) \tag{4}$$

2.2 Weak Constraints

A weak constraint has the form

$$:\sim l_1,\ldots,l_n.\ [weight@level] \tag{5}$$

where ls are literals, *weight* is a real number and *level* is an integer. Let Π be an ASP program $\Pi_1 \cup \Pi_2$, where Π_1 is a set of rules of the form (1) and Π_2 is a set of weak constraints. We call X a stable model of Π if it is a stable model of Π_1. For a stable model X of Π, the *penalty* of X at level l, denoted by $penalty(\Pi,X,l)$, is defined as

$$penalty(\Pi,X,l) = \sum_{X\not\models\ :\sim l_1,\ldots,l_n.\ [w@l]} w \tag{6}$$

For any two stable models X_1 and X_2 of Π, we say X_1 is dominated by X_2 if

- there is some integer l such that $penalty(\Pi,X_2,l) < penalty(\Pi,X_1,l)$ and
- for all integers $k > l$, $penalty(\Pi,X_2,k) = penalty(\Pi,X_1,k)$.

A stable model of Π is *optimal* if it is not dominated by another stable model of Π.

3 Partition

In this section, we first present the concept of augmented subset and some related definitions that are the theoretic foundation of the partition stage of the solver. Then, we present a partition algorithm.

3.1 Augmented Subsets and Related Notations

Definition 1 (Augmented Subset). *For a ground LP^{MLN} program M, an augmented subset of M is a triple tuple $S = (I, SAT, UNS)$, where I, SAT and UNS are three pairwise disjoint subsets of M that satisfy $I \cup SAT \cup UNS = M$.*

For example, let $w : r$ be a rule of a ground LP$^{\text{MLN}}$ program M, $(M - \{w : r\}, \emptyset, \{w : r\})$ and $(M, \emptyset, \emptyset)$ are both augmented subsets of M.

Definition 2 (Stable Models of an Augmented Subset). *For an augmented subset $S = (I, SAT, UNS)$ of a ground LP^{MLN} program M, a set X of ground literals is a stable model of S iff X is a stable model of I that satisfy SAT and does not satisfy any rule in UNS.*

From the definition of stable models of an augmented subset, we can observe that the satisfiability of rules in set SAT and UNS are determinate, while the satisfiability of rules in set I are indeterminate. Hence, the rules in I are called indeterminate rules, and the rules in SAT and UNS are called determinate rules. By $SM'(S)$, we denote all stable models of an augmented subset S of an LP$^{\text{MLN}}$ program M. For a set $X \in SM'(S)$, the weight degree $W'(S, X)$ of X w.r.t. S is defined as

$$W'(S, X) = exp\left(\sum_{w:r \in I_X \cup SAT} w\right) \tag{7}$$

and the weight degree $W'_p(S, \beta)$ of a proposition β w.r.t. S is defined as

$$W'_p(S, \beta) = \sum_{X \in SM'(S) \text{ and } X \models \beta} W'(S, X) \tag{8}$$

Definition 3 (Split). *For an LP^{MLN} program M and an augmented subset S of M, a set $SP = \{S_i \mid 1 \leq i \leq n\}$ of augmented subsets of M is called a split of S, iff*

$$SM'(S) = \bigcup_{i=1}^{n} SM'(S_i) \tag{9}$$

where $SM'(S_i) \cap SM'(S_j) = \emptyset$ for any $i \neq j$, and for any stable model $X \in SM'(S_i)$

$$W'(S, X) = W'(S_i, X) \tag{10}$$

where $1 \leq i \leq n$.

Suppose $S_1 = (I_1, SAT_1, UNS_1)$ and $S_2 = (I_2, SAT_2, UNS_2)$ are two augmented subsets of an LP$^{\text{MLN}}$ program M, and $w : r$ is a rule in M, we say S_1 and S_2 are complementary on the rule $w : r$ iff (1) $I_1 = I_2$, and (2) $SAT_1 - SAT_2 = \{w : r\}$(or $SAT_2 - SAT_1 = \{w : r\}$). In addition, if S_1 and S_2 are complementary on a rule $w : r$, then we can define the *union* of two augmented subsets $S_1 \sqcup S_2 = (I_1 \cup \{w : r\}, SAT_1 - \{w : r\}, UNS_1 - \{w : r\})$. For example, suppose $S_1 = (M - \{w : r\}, \emptyset, \{w : r\})$, and $S_2 = (M - \{w : r\}, \{w : r\}, \emptyset)$, then S_1 and S_2 are complementary on the rule $w : r$, and $S_1 \sqcup S_2 = (M, \emptyset, \emptyset)$.

Definition 4 (Substitute). *For an LP^{MLN} program M, a set ST of augmented subsets of M is called a substitute of an augmented subset S of M with regard to a rule $w : r$ if ST contains only two elements S_1 and S_2 such that S_1 and S_2 are complementary on the rule $w : r$ and $S_1 \sqcup S_2 = S$.*

Lemma 1. *For a ground LP^{MLN} program M and an augmented subset S of M, a substitute $ST = \{S_1, S_2\}$ of S is a split of S.*

Proof. By the definition of split, the proof of Lemma 1 has three parts. Firstly, by the definition of substitute, it is easy to show that $SM'(S_1) \cap SM'(S_2) = \emptyset$. Secondly, by the definition of substitute and stable models of augmented subsets, we can show that $SM'(S) = SM'(S_1) \cup SM'(S_2)$. Finally, by the definition of weight degrees of stable models w.r.t. an augmented subset, we can show that $W'(S, X) = W'(S', X)$, where $S' \in \{S_1, S_2\}$ and $X \in SM'(S')$ (the complete proofs of all lemmas and theorems can be found in the extended version of the paper [1]). $\square$

Lemma 2. *For an LP^{MLN} program M and the augmented subset $S_0 = (M, \emptyset, \emptyset)$ of M, stable models and their weights of S_0 and M coincide, which means $SM(M) = SM'(S_0)$ and $W(M, X) = W'(S_0, X)$ for any $X \in SM(M)$.*

Lemma 2 can be easily proven by the definition of stable models and their weights of LP$^{\text{MLN}}$ programs and augmented subsets. Combining with Lemma 1, Lemma 2 provides a way to compute stable models and their weights of an LP$^{\text{MLN}}$ program M via computing them of augmented subsets in a split w.r.t. M, which can be done concurrently.

In addition, probability degrees of a stable model and a proposition defined by Equation (3) and (4) can be restated in terms of split. For an LP$^{\text{MLN}}$ program M, a split SP of $(M, \emptyset, \emptyset)$, an augmented subset $S_i \in SP$, and a stable model $X \in SM'(S_i)$, the probability degree of X w.r.t. M can be restated as follows:

$$P_s(M, X) = \lim_{\alpha \to \infty} \frac{W'(S_i, X)}{\sum_{S_j \in SP} \sum_{X' \in SM'(S_j)} W'(S_j, X')} \tag{11}$$

and the probability degree of a proposition β w.r.t. M can be restated as follows:

$$P_p(M, \beta) = \lim_{\alpha \to \infty} \frac{\sum_{S_k \in SP} W'(S_k, \beta)}{\sum_{S_j \in SP} \sum_{X' \in SM'(S_j)} W'(S_j, X')} \tag{12}$$

Let us recall two inference tasks supported by our solver in the view of split. For the MAP task, by the definition of split, we can find the most probable stable models of augmented subsets in a split, then find the most probable stable models of input program among those of augmented subsets. For the PI task, we can observe that part of Equation (11) and (12) can be evaluated from an augmented subset, which can be used to simplify the computing in the Synthesis stage. Hence both MAP task and PI task of an LP$^{\text{MLN}}$ program can be solved concurrently in the sense of split.

[1] http://cse.seu.edu.cn/PersonalPage/seu_zzz/publications/parallel-lpmln-solver.pdf

3.2 Partition Algorithm

Inference tasks of an LP$^{\mathrm{MLN}}$ program M can be reduced to tasks of the augmented subset $S_0 = (M, \emptyset, \emptyset)$, where S_0 can be partitioned into a split, which laid the theoretic foundation of our parallelized method for solving the program M. In this section, we present a method to obtain a split.

Definition 5 (Transformer). *For an LP^{MLN} program M, a set T of augmented subsets of M is called a transformer of M if there exists a finite sequence of sets of augmented subsets $T^1, T^2, \ldots, T^n$ satisfying*

- *$T^{i+1} = (T^i - \{S_k\}) \cup ST_k$, and*
- *$T^1 = \{(M, \emptyset, \emptyset)\}$ and $T^n = T$.*

where $S_k \in T^i$ and ST_k is a substitute of S_k.

Theorem 1. *For an LP^{MLN} program M, if a set T of augmented subsets of M is a transformer of M, then T is a split of the augmented subset $S_0 = (M, \emptyset, \emptyset)$.*

Proof. By Lemma 2 and Lemma 1, Theorem 1 can be proved by mathematical induction. □

Theorem 1 leads to a method to partition an augmented subset of an LP$^{\mathrm{MLN}}$ program into a split. The process showed in Algorithm 2 is an implementation of the method.

Algorithm 2: Partition

```
  Input: $M$: an LP$^{\mathrm{MLN}}$ program, $n$: size of the split
  Output: a split $SP$ of the augmented subset $(M, \emptyset, \emptyset)$
1 begin
2     $SP = \{(M, \emptyset, \emptyset)\}$;
3     while $|SP| \leq n$ do
4         select an augmented subset $S_k \in SP$ randomly;
5         select a substitute $ST_k$ of $S_k$ randomly;
6         $SP = (SP - \{S_k\}) \cup ST_k$ ;
7     return $SP$;
```

4 Translation

Inspired by the translation introduced by Lee and Yang in [13], we propose a new translation from augmented subsets of an LP$^{\mathrm{MLN}}$ program M to ASP programs such that stable models and associated weights of an augmented subsets can be derived from the translation. And we also follow the point of [5] by identifying logic program rules as a special case of first order formulas under the stable model semantics. Hence, an LP$^{\mathrm{MLN}}$ program can be viewed as a set of weighted formulas. Here, we first review the translation given in [13].

4.1 Lee and Yang's Translation

For an LP$^{\text{MLN}}$ program M, the translation $\tau(M)$ is defined as follows. Every weighted formula $w : F$ is translated into a choice formula "$F \vee \neg F$" and a weak constraint ":$\sim$ F. [$weight@lev$]", where $weight = -1$ and $lev = 1$ if w is α, and $weight = -w$ and $lev = 0$ otherwise. It has been shown that the stable models of M are exactly the stable models of $\tau(M)$, and the most probable stable models of M are precisely the optimal stable models of $\tau(M)$.

Although the translation is designed for solving the MAP task for an LP$^{\text{MLN}}$ program, it can also be used for computing the weight for a stable model. For a stable model X of $\tau(M)$, the sum of weights of formulas satisfied by X can be derived from the penalties of X, that is

$$sum(M, X) = penalty(\tau(M), X, 1) * \alpha + penalty(\tau(M), X, 0) \tag{13}$$

The sum is an intermediate result for computing weight of X in the sense of LP$^{\text{MLN}}$, and the weight degree of X can be restated as follows:

$$W(M, X) = exp(-sum(M, X)) \tag{14}$$

Unfortunately, Lee and Yang's translation is not enough for our situation. For an LP$^{\text{MLN}}$ program M, an augmented subset $S = (I, SAT, UNS)$ of M consists of three kinds of formulas. The formulas in I are ordinary LP$^{\text{MLN}}$ formulas, which can be translated by using Lee and Yang's method, while the formulas in SAT and UNS are determinate formulas, which need to be translated by using new translating method. Our new translation extends Lee and Yang's translation and completely captures the stable models and their penalties of augmented subsets of an LP$^{\text{MLN}}$ program.

4.2 New Translation

For an augmented subset $S = (I, SAT, UNS)$ of a ground LP$^{\text{MLN}}$ program M, the translation $\tau'(S)$ of S is defined as follows. Firstly, we turn each weighted formula $w_k : F_k$ in I to $F_k \vee \neg F_k$. Secondly, we turn each weighted formula $w_k : F_k$ in SAT to F_k. Finally, we turn each weighted formula $w_k : F_k$ in UNS to $\neg F_k$.

Furthermore, we use weak constraints to compute the sum of weighted formulas satisfied by a stable model, which is slightly different from Lee and Yang's translation. For each weighted formula $w_k : F_k$ in I, we add the following formulas into our translation $\tau'(S)$:

$$F_k \rightarrow sat(k) \tag{15}$$

$$\neg sat(k) \rightarrow \neg F_k \tag{16}$$

where $sat(k)$ is a fresh atom, and if $w_k : F_k$ is a hard formula, add a weak constraint

$$:\sim\ sat(k).\ [1@1] \tag{17}$$

and if $w_k : F_k$ is a soft formula, add a weak constraint

$$:\sim sat(k).\ [w_k@0] \tag{18}$$

Theorem 2. *For any augmented subset $S = (I, SAT, UNS)$ of a ground LP^{MLN} program M, the set $SM'(S)$ and the set of stable models of $\tau'(S)$ coincide on the literals of M. And for a stable model X in $SM'(S)$, the weight of X can be computed by*

$$W'(S, X) = exp(sum(I, X) + \sum_{w:r \in SAT} w) \tag{19}$$

Theorem 2 tells us that an augmented subset S of a ground LP$^{\text{MLN}}$ program M can be solved by solving its translation $\tau'(S)$, and the weight of a stable model X of S can be computed by computing penalties of X. And the theorem can be proved by the Proposition 1 from [13].

5 Experiments

5.1 Test Environment and Test Cases

We tested our implementation on a Dell PowerEdge R920 server with an Intel Xeon E7-4820@2.00GHz with 32 cores and 24 GB RAM running the Ubuntu 16.04 operating system. We used Clingo 4.5.4 as our back-end ASP solver, and F2LP [11] to translate first order formulas into ASP programs.

We used two problems: the Monty Hall problem and the Bird problem as our test cases, where hard rules of corresponding programs are treated as determinately satiable rules in this paper (the source code of the experiments can be found on GitHub[2]). The Bird problem and its LP$^{\text{MLN}}$ program are from Example 1 in [12]. Following the description of the Monty Hall Problem from [19], we encoded it in LP$^{\text{MLN}}$ as below.

1. One of three boxes contains a key to a car, and the possibility of each box containing the key is the same.

$$1\ :\ has\_key(X) \leftarrow box(X). \tag{20}$$

   In the Monty Hall Problem with three boxes, ground rules of the rule (20) are the following three rules. The weights of all these rules are the same, which expresses the possibility degrees of all boxes containing the key are the same. Actually, the value of the weight of the rule (20) can be any real number.

$$1 : has\_key(1) \leftarrow box(1).$$
$$1 : has\_key(2) \leftarrow box(2).$$
$$1 : has\_key(3) \leftarrow box(3).$$

[2] https://github.com/wangbiu/lpmlnmodels

2. The player selects one of boxes randomly.

$$1 \ : \ \mathit{select}(X) \leftarrow \mathit{box}(X). \tag{21}$$

3. The host opens one of unselected boxes that does not contain the key randomly.

$$1 : \mathit{open}(X) \leftarrow \mathit{box}(X), \mathit{not}\ \mathit{cannot\_open}(X). \tag{22}$$
$$\mathit{cannot\_open}(X) \leftarrow \mathit{select}(X). \tag{23}$$
$$\mathit{cannot\_open}(X) \leftarrow \mathit{has\_key}(X). \tag{24}$$

4. If the player changes his mind, he switches to remaining boxes randomly.

$$1 : \mathit{switch}(X) \leftarrow \mathit{can\_switch}(X). \tag{25}$$
$$\mathit{can\_switch}(X) \leftarrow \mathit{box}(X), \mathit{not}\ \mathit{select}(X), \mathit{not}\ \mathit{open}(X). \tag{26}$$

5. The winning rules under two cases are encoded by *win_stay* and *win_switch*.

$$\mathit{win\_stay} \leftarrow \mathit{select}(X), \mathit{has\_key}(X). \tag{27}$$
$$\mathit{win\_switch} \leftarrow \mathit{switch}(X), \mathit{has\_key}(X). \tag{28}$$

6. Finally, we encode that there are three boxes, and we can increase the number of boxes.

$$\mathit{box}(1). \quad \mathit{box}(2). \quad \mathit{box}(3). \ \ldots$$

Additionally, there are some underlying restrictions: (a) the player can select only one box, (b) the host can open only one box too, (c) only one of those boxes contains the key, and (d) the player can switch his choice only once. These restrictions are encoded by following definite rules.

$$\neg \mathit{select}(Y) \leftarrow \mathit{select}(X), \mathit{box}(Y), X \neq Y. \tag{29}$$
$$\leftarrow \mathit{box}(X), \mathit{not}\ \mathit{select}(X), \mathit{not}\ \ \neg \mathit{select}(X). \tag{30}$$
$$\neg \mathit{open}(Y) \leftarrow \mathit{open}(X), \mathit{box}(Y), X \neq Y. \tag{31}$$
$$\leftarrow \mathit{box}(X), \mathit{not}\ \mathit{open}(X), \mathit{not}\ \ \neg \mathit{open}(X). \tag{32}$$
$$\neg \mathit{has\_key}(Y) \leftarrow \mathit{has\_key}(X), \mathit{box}(Y), X \neq Y. \tag{33}$$
$$\leftarrow \mathit{box}(X), \mathit{not}\ \mathit{has\_key}(X), \mathit{not}\ \ \neg \mathit{has\_key}(X). \tag{34}$$
$$\neg \mathit{switch}(Y) \leftarrow \mathit{switch}(X), \mathit{can\_switch}(Y), X \neq Y. \tag{35}$$
$$\leftarrow \mathit{can\_switch}(X), \mathit{not}\ \mathit{switch}(X), \mathit{not}\ \neg \mathit{switch}(X). \tag{36}$$

In both problems, we can increase the number of the boxes or the birds to increase the size of problems. By *montyN* and *birdN*, we denote the Monty Hall problem with N boxes and the Bird problem with N birds, respectively. In our experiments, MAP task is to output all most probable stable models, and PI task is to output all literals and their probabilities. And the programs are grounded manually due to the lack of LP$^{\mathrm{MLN}}$ grounder.

5.2 Test Results

In our experiments, each running time was the average of five tests. Firstly, we tested our implementation in a non-parallel mode. Table 1 shows that ASP solver accounts for most of the running time in solving an LP$^{\text{MLN}}$ program, and the running times of solving the problems increase as the sizes of the problems increase.

Table 1. Running Time (seconds) in non-parallel mode

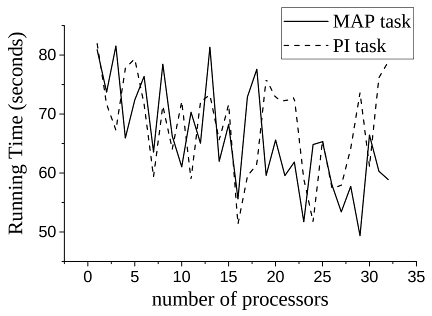
(a) the Monty Hall problem

M	MAP		PI	
	clingo	total	clingo	total
*monty*3	0.01	0.05	0.02	0.04
*monty*10	0.14	0.22	0.18	0.26
*monty*20	2.33	2.59	2.30	2.78
*monty*30	17.63	18.28	16.97	18.04
*monty*40	76.19	77.54	77.00	79.87

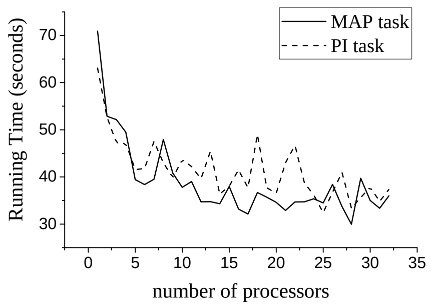
(b) the Bird problem

M	MAP		PI	
	clingo	total	clingo	total
*bird*4	0.01	0.02	0.01	0.01
*bird*8	0.04	0.09	0.07	0.10
*bird*10	0.42	0.46	0.40	0.57
*bird*12	4.30	4.53	4.03	4.96
*bird*14	53.08	57.42	48.76	59.65

Secondly, we tested our implementation in a parallel mode. Figure 1 shows that running times of solving the *bird*14 problem are on the decline with an increase of the number of processors used, while running times of solving the *monty*40 problem shows an obvious fluctuation. But we can find that the general trend appears to be decrease.



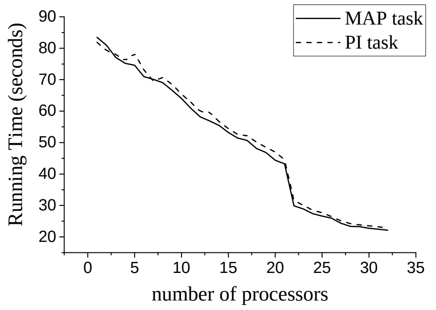
(a) program *monty*40



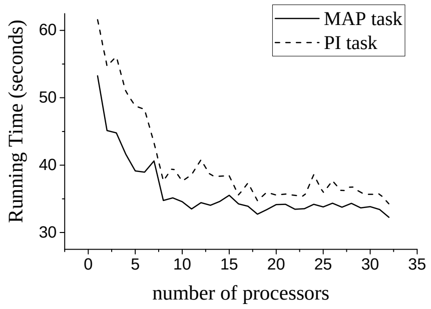
(b) program *bird*14

Fig. 1. Running Time vs. number of processors used

Finally, we tested our parallel solver by adding some heuristic information. In our experiments, we tried to avoid contradictions in an augmented subset and keep every augmented subset containing the same numbers of indeterminate rules. For example, rules "$0 : a$" and "$0 : \neg a$" are contradictory, hence, we do not put both of them in the set of determinately satiable rules of an augmented subset. For the Monty Hall problem, rules of the form (30), (32), (34), (36) and other corresponding rules may lead to contradictions. We can see from Figure 2 that running times of both programs show a more steady improvement with the increase of processors used.



(a) program $monty40$



(b) program $bird14$

Fig. 2. Running Time vs. number of processors used (heuristic partition)

5.3 Discussion

Our data suggests that solving a split of an LP$^{\text{MLN}}$ program can improve the efficiency of solving the LP$^{\text{MLN}}$ program. The results in Table 1 illustrate that the more indeterminate rules an LP$^{\text{MLN}}$ program contains, the harder solving the program will be. Hence, by decreasing numbers of indeterminate rules of an augmented subset, our parallel solver can make an augmented subset more easy to solve.

The results in Figure 2 and Figure 1 also illustrate that the partition method in our parallel solver is specially important for the parallelized solving, and the random partition method used in the paper is not a good choice. A possible reason is that Algorithm 2 cannot always generate a balanced split w.r.t. an input program. A balanced split w.r.t. an LP$^{\text{MLN}}$ program means every augmented subset of the split is solved by taking approximately the same length of time. For now, the partition method to generate a balanced split still remains unclear, hence, it is worthy to find such a method.

6 Conclusion and Future Work

We implemented a parallel LP$^{\text{MLN}}$ solver, which computes the stable models as well as their weights in a parallel way. And we presented some theoretic discussion on the implementation containing the partition method and the translation method. Experimental results show that our parallelized method can improve performance of the LP$^{\text{MLN}}$ solver. Besides, we made a preliminary attempt on how different partition methods influence our parallelized solving. Experimental results show that the partition method using some heuristic information is better than the stochastic method on our two test cases.

For the future, we plan to make an elaborative investigation on heuristic partition methods to make our parallel solver faster. And we also plan to test our solver further by modeling with LP$^{\text{MLN}}$ in more real-world applications.

7 Acknowledgments

We are grateful to the anonymous referees for their useful comments. This work was supported by the National High Technology Research and Development Program of China (Grant No.2015AA015406), the National Science Foundation of China (Grant No.60803061), the National Science Foundation of China (Grant No.61272378), and the Natural Science Foundation of Jiangsu (Grant No.BK2008293).

References

1. Balai, E., Gelfond, M.: On the Relationship between P-log and LP$^{\text{MLN}}$. In: Proceedings of the 25th International Joint Conference on Artificial Intelligence. (2016) 915–921
2. Baral, C., Gelfond, M., Rushton, N.: Probabilistic reasoning with answer sets. Theory and Practice of Logic Programming **9**(1) (2009) 57–144
3. Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Ricca, F., Schaub, T.: ASP-Core-2 Input language format. (2012)
4. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic prolog and its application in link discovery. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence. Volume 7. (2007) 2468–2473
5. Ferraris, P., Lee, J., Lifschitz, V.: Stable models and circumscription. Artificial Intelligence **175**(1) (2010) 236–263
6. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: CLASP: A Conflict-Driven Answer Set Solver. In: Logic Programming and Nonmonotonic Reasoning. (2007) 260–265
7. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In: Logic Programming, Proceedings of the Fifth International Conference and Symposium. (1988) 1070–1080
8. Kok, S., Sumner, M., Richardson, M., Singla, P., Poon, H., Lowd, D., Wang, J., Domingos, P.: The Alchemy System for Statistical Relational AI. Technical report, Department of Computer Science and Engineering, University of Washington, Seattle, WA. (2009)
9. Lee, J., Lifschitz, V.: Loop Formulas for Disjunctive Logic Programs. In: Logic Programming, 19th International Conference. (2003) 451–465
10. Lee, J., Meng, Y., Wang, Y.: Markov logic style weighted rules under the stable model semantics. In: Proceedings of the Technical Communications of the 31st International Conference on Logic Programming. (2015)
11. Lee, J., Palla, R.: System f2lp Computing Answer Sets of First-Order Formulas. In: Logic Programming and Nonmonotonic Reasoning: 10th International Conference, LPNMR 2009, Potsdam, Germany, September 14-18, 2009. Proceedings. Springer Berlin Heidelberg (2009) 515–521
12. Lee, J., Wang, Y.: Weighted Rules under the Stable Model Semantics. In: Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, AAAI Press (2016) 145–154
13. Lee, J., Yang, Z.: LP$^{\text{MLN}}$, Weak Constraints, and P-log. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence. (2017)

14. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. Acm Transactions on Computational Logic **7**(3) (2006) 499–562
15. Lin, F., Zhao, Y.: ASSAT: Computing answer sets of a logic program by SAT solvers. Artificial Intelligence **157**(1-2) (2004) 115–137
16. Niu, F., Ré, C., Doan, A., Shavlik, J.: Tuffy: Scaling up Statistical Inference in Markov Logic Networks using an RDBMS. Proceedings of the VLDB Endowment **4**(6) (2011) 373–384
17. Pearl, J.: Causality: Models, reasoning, and inference. **29** (2000)
18. Richardson, M., Domingos, P.M.: Markov logic networks. Machine learning **62**(1-2) (2006) 107–136
19. Selvin, S.: A problem in probability (letter to the editor). The American Statistician **29**(1) (1975) 67