Ph.D. Qualifying Exam Presentation

Supervisor:	Mirosław Truszczyński
Committee Members:	Raphael Finkel
	Carl Lee
	Victor Marek
	Mirosław Truszczyński
Student:	Lengning Liu

Department of Computer Science

University of Kentucky

February 24th, 2003

Goals for the Presentation

- **Provide a broad overview of my research area**
- Introduce relevant technical background
- Describe logic $PS+$ - a focus for my research
- Present my current work
- Discuss research directions to pursue

Overview

- General area of study: logic in programming and computing
- Specific focus: answer-set programming (ASP)
  - ASP emerged in late 90’s
  - theories encode problems so that models represent solutions
  - to program - we need a language
  - to compute - we need algorithms
- Goal: to develop fast programs to support ASP formalism based on logic PS+

Overview (contd.)

- ASP is in contrast with traditional use of logic in programming and computing
  - automated theorem proving (50’s and 60’s)
  - logic programming (70’s PROLOG)
  - problems are encoded as queries to theories so that proofs and variable substitutions determine solutions

Goals for the Presentation

- Provide a broad overview of my research area $\surd$
- **Introduce relevant technical background**
- Describe logic $PS+$ - a focus for my research
- Present my current work
- Discuss research directions to pursue

Technical Background

- **Propositional logic**
  - Basis for defining semantics of logic programs
  - Basis for computing models of logic programs
- **First order logic**
  - Provides programming facilities
  - Separates data from programs
- Logic programming
- Default logic

Propositional Logic - Language

- Basic Syntax:
  - Atoms: $a, b, c, \ldots$
  - Boolean connectives: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$
  - Parentheses: $(,)$
- Formulas: $(a \vee b) \rightarrow \neg c$
- Literals: atoms and their negations: a, $\neg a$
- Clauses: disjunctions of literals: $a \vee b \vee \neg c$
- Theory: a collection of formulas
- CNF theories (main focus): a collection of clauses

Propositional Logic - Semantics

- Interpretation: assignment I of truth value **t** or **f** to atoms
  - often represented as the set of all atoms assigned **t**
  - all others, by default, assigned **f**
  - formulas obtain truth value in an inductive way under I
- Model: interpretation I is a model of a theory if every formula in the theory obtains **t** under I

SAT Problems

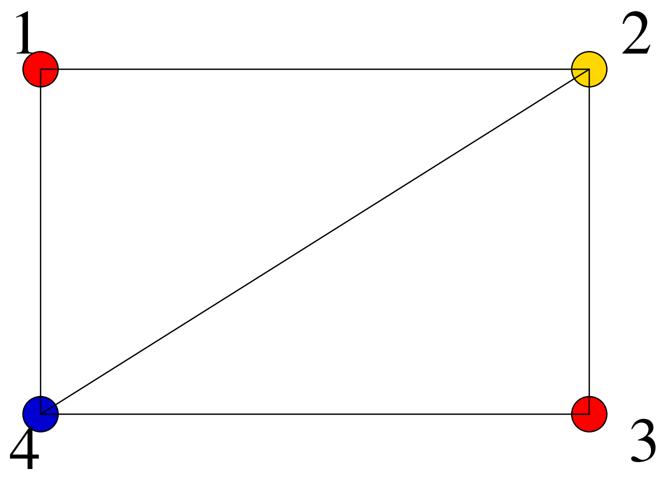
- Problem: Given a propositional CNF theory φ, decide whether it has a model.
- It is a prototypical NP-complete problem.
- We often want not only a decision but also a witness (in case the answer is YES)
- We sometimes want all models
- SAT solvers are programs that compute models of a propositional CNF theory
- Propositional logic is an ASP formalism
  - We can encode problems by propositional CNF theories and use SAT solvers to find solutions for the problems

Example - 3 Colorability Problem

- Instance: An undirected graph and 3 colors
- Question: Can we color the graph such that
  1. each vertex gets exactly one color;
  2. no two adjacent vertices have the same color;
- Example Graph: A graph with 4 vertices and 5 edges

Example - 3 Colorability Problem

- Instance: An undirected graph and 3 colors
- Question: Can we color the graph such that
  1. each vertex gets exactly one color;
  2. no two adjacent vertices have the same color;
- Example Graph: A graph with 4 vertices and 5 edges



CNF Encoding for 3-Col

% Each vertex gets at least one color

$colored_{i1} \vee colored_{i2} \vee colored_{i3}$

% Each vertex gets at most one color

$\neg colored_{i1} \vee \neg colored_{i2}$

$\neg colored_{i2} \vee \neg colored_{i3}$

$\neg colored_{i3} \vee \neg colored_{i1}$

% No two adjacent vertices have the same color

$\neg colored_{i1} \vee \neg colored_{j1}$

$\neg colored_{i2} \vee \neg colored_{j2}$

$\neg colored_{i3} \vee \neg colored_{j3}$

Properties of the Encoding

- Theorem:
  - the encoding in the previous slide correctly encodes the 3-colorability problem on the given graph;
  - models of the encoding and the proper coloring schemas have a one-to-one correspondence.
- The encoding is not flexible because there is no separation on data and program.

Solvers for SAT Problems

- To get solutions of the problem – we need to find models of the CNF theory that encodes the problem
- There are two types of solvers:
  - Complete Solvers
    - use a systematic search schema to examine the whole search space
    - find and output models if the instance is satisfiable; output “no” otherwise;
  - Incomplete Solvers
    - use stochastic local search schema
    - are not guaranteed to find models if there are any, and will not output "no" when the instances are unsatisfiable

Systematic Search Schema

- Basic Search Schema
  - Use backtrack search
    - Extend current partial interpretation by assigning a truth value to a new atom;
    - Examine if there are any conflicts caused by the current partial interpretation;
    - Undo the most recent assignment to the atom if there are unsatisfied clauses, and trying the opposite assignment to that atom, if it has not yet been tried.
- Optimizations
  - Constraint propagation
  - Learning from failure
  - Backjumping

Local Search Schema

- Generate initial truth assignment randomly
- Walk from one truth assignment to another by flipping one atom at a time
- Use greedy algorithm to find local minima
- Use randomized techniques to escape from local minima or plateau

Generic Local Search Algorithm

$Generic\text{-}Local\text{-}Search(F)$

`Input:`

- F `- a propositional CNF formula`

`Output:`

- `a satisfying assignment of` F `if it can be found`

Generic Local Search Algorithm (contd.)

```
BEGIN
1.   For i ← 1 to MAX-TRIES, do
2.         σ ← randomly generated truth assignment;
3.         For j ← 1 to MAX-FLIPS, do
4.               If σ ⊨ F, return σ;
5.               a ← Pick-Atom(F, σ);
6.               (F, σ) ← Flip(F, σ, a);
7.         End for
8.   End for
END
```

Two Families of Local Search Algorithms

- GSAT Family: watch all atoms in the theory
  - GSAT: flips the variable that minimizes the total number of unsat clauses;
  - GSAT-SA: uses simulated annealing algorithm to escape from local minima or plateaus;
  - GSAT-RW: with probability p, it selects an unsat clause and flip one of the variables; with probability $1 - p$, it follows GSAT;
  - GSAT-RW-TABU: keeps a FIFO list of flipped variables of fixed length and forbids any of the variables in the list to be flipped again;

Two Families of Local Search Algorithms Contd.

- WSAT Family: pick an unsat clause and only watch atoms in that clause
  - WSAT-G: with probability p flips any variable, otherwise, flips the one that minimizes the total number of unsat clauses;
  - WSAT-B: with probability p flips any variable, otherwise, flips the one that causes the least number of sat clauses to become unsat (i.e. the least $break$-$count$);
  - WSAT-Free: if there is a variable such that none of the clauses will become unsat if it is flipped, then flips it; otherwise, follows WSAT-B;

GSAT

$Pick\text{-}Atom\text{-}GSAT(F, \sigma)$

```
BEGIN
1.  For each atom x in F, do
2.        σ' ←the truth assignment that differs from σ in x;
3.        u_1 ←the number of unsatisfied clauses in F under σ;
4.        u_2 ←the number of unsatisfied clauses in F under σ';
5.        Δw(x) ← u_2 − u_1;
6.  End for
7.  return argmin_x({Δw(x) : Δw(x) ≤ 0}) or nothing if such x does not exist;
END
```

WalkSAT-Free

$Pick\text{-}Atom\text{-}WalkSAT\text{-}Free(F)$

BEGIN
1. $C \leftarrow$ randomly selected unsatisfied clause;
2. For each atom x in C, compute $break\text{-}count(x)$;
3. If there are any atoms that have zero break-count, return any one of them;
4. With probability p, return $argmin_x\{break\text{-}count(x)\}$;
5. With probability $1-p$, return a randomly chosen atom in C;
END

First-order Logic - Language

- Basic syntax
  - Variable symbols: $x, y, z, \cdots$
  - Function symbols: $f, g, h, \cdots$
  - Relation symbols: $p, q, r, \cdots$
  - Logic connectives: $\wedge, \vee, \neg, \rightarrow, \leftrightarrow$
  - Quantifiers: $\forall, \exists$
  - Parentheses: $(,)$
- Terms: $f(t_1, \ldots, t_n)$
- Formulas: $r(t_1, \ldots, t_n) \wedge (p(s_1, \ldots, s_m) \vee \neg q(u))$
- Theories: a set of formulas

First-order Logic - Semantics

- Interpretation (Model) I:
  - Nonempty universe: A
  - Interpretation of function symbols: $f^I : A^n \to A$
  - Interpretation of relation symbols: $r^I \subseteq A^n$
- Truth value of a formula under an interpretation I
  - $\forall x \varphi$ obtains value **t** if $\varphi\{x/a\}$ obtains value **t** for every $a \in A$

First-order Logic - Herbrand Models

- We often focus on Herbrand models because
  - they define natural universes
  - a universal sentence has a model iff it has a Herbrand model
- Define a Herbrand model as follows:
  - Common parts for all Herbrand models w.r.t. a fixed first-order theory
    - Herbrand universe: $a, f(a), f(f(a)), \cdots$
    - Herbrand base: $r(a), r(f(a)), r(f(f(a))), \cdots$
  - Difference between Herbrand models
    - Interpretation of relation symbols

First-order Logic - Grounding

- A first-order formula is a **universal sentence** if it is of the following form:
  - $\forall x_1 \cdots \forall x_n \varphi$, where φ has no quantifiers and $x_1, \ldots, x_n$ are all the variables that occur in φ
- A substitution θ is a **ground substitution** of a formula φ, if $theta$ maps all free variables in φ to ground terms
- Let φ be a universal sentence, $\varphi\theta$ is a **ground instance** of φ if θ is a ground substitution of φ
- Let φ be a universal sentence, $\mathbf{ground}(\varphi)$ denotes the set of all ground instances of φ
- Let P be a set of universal sentences, $\mathbf{ground(P)}$ denotes the union of all $ground(\varphi)$ for every $\varphi \in P$

First-order Logic - Grounding (contd.)

- Theorem: P is a set of universal sentences, the following are equivalent:
  - P has a model
  - P has a Herbrand model
  - $ground(P)$ is satisfiable
- Theorem: The set of Herbrand models of $ground(P)$ is the same as the set of Herbrand models of P

Semantics via Grounding

- To define the notion of an "intended model" in some formalism based on the language of first-order language:
  - restrict to Herbrand models
  - define "intended models" for a propositional fragment of the formalism
  - lift the semantics to the general case through grounding
  - generic definition: M is an "intended model" of T if M is an "intended model" of $ground(T)$

Goals for the Presentation

- Provide a broad overview of my research area $\surd$
- Introduce relevant technical background $\surd$
- **Describe logic $PS+$ - a focus for my research**
- Present my current work
- Discuss research directions to pursue

Logic of Propositional Schemata

- Syntax of logic PS is a fragment of first-order language:
  - infinite denumerable sets R, C and V of relation, constant and variable symbols;
  - symbol $\perp$ and $\top$ (always interpreted as **f** and **t**);
  - boolean connectives $\wedge$, $\vee$ and $\rightarrow$, the universal and existential quantifiers, and punctuation symbols '$($', '$)$' and '$,$'.

E-Atoms and Formulas in Logic PS

- $\exists X_1, \ldots, X_k p(t_1, \ldots, t_n)$
  is an e-atom. As an abbreviation, variables $X_1, \ldots, X_k$ can be expressed by '_' (underscore).
  - $p(\_, X)$
- The only allowed formulas in the logic PS are rules, which are of the following form:
  $\forall X_1, \ldots, X_k (A_1 \wedge \cdots \wedge A_m \rightarrow B_1 \vee \cdots \vee B_n)$
  and e-atoms are allowed only in $B_i's$.
  - $q(X) \rightarrow p(\_, X)$.

Example of Grounding

- Given a rule $\forall X q(X) \rightarrow p(\_, X)$., and the Herbrand universe $\{a, b, c\}$,
  - Ground of e-atom $p(\_, X)$ is

$$p(a, X) \vee p(b, X) \vee p(c, X)$$

  - An instance of the rule is

$$q(a) \rightarrow p(a, a) \vee p(b, a) \vee p(c, a)$$

  - There are three such instances
  - Ground theory of the rule consists of all three instances

Logic PS+

- Cardinality constraints are common in combinatorial problems: size of the vertex cover is at most 5;
- C-atoms are used to model cardinality constraints
- The resulting logic is called logic $PS+$

Cardinality Atoms

- A **ground cardinality atom** (or a c-atom) is of the following form:

$$m\{a_1, ..., a_k\}n$$

  where a_i's are atoms and m, n are two non-negative integers
- The c-atom obtains truth value **t** under an interpretation I if there are at least m and at most n atoms out of $a_1, ..., a_k$ obtain **t** under I.

Logic PS+ Theory

- Logic $PS+$ **theories** are defined by data-program pairs (D, P)
- D is a finite set of ground atoms
- P is a finite set of $PS+$ rules.
- $Cl(D)$ denotes the theory $D \cup \{\neg a : a \notin D\}$
- The semantics is given by the set of Herbrand models of $Cl(D) \cup ground(P)$

Logic PS+ as an ASP Formalism

- Encode the problem into a data-program pair (D, P)
  - D - encoding of relevant input data
  - P - declarative specification of the computational task
- Ground P to $ground(P)$ using a grounder
- Compute models of $Cl(D) \cup ground(P)$ using a solver
- Reconstruct solutions from models
- Logic PS+ fits into the answer-set programming paradigm, where models of a logic program correspond to solutions of the problem that the program encodes

Logic PS+ Encoding for 3-Col

- Define **data** part

  % Define colors and the graph

  ```
  color(red).  color(blue).  color(green).
  vtx(1).  vtx(2).  vtx(3).  vtx(4).
  edge(1,2).  edge(2,3).  edge(3,4).  edge(4,1).
  edge(2,4).
  ```

- Define **program** part

  % Typing constraints

  colored(X, C) $\rightarrow$ vertex(X).

  colored(X, C) $\rightarrow$ color(C).

  % Each vertex get exactly one color

  vertex(X) $\rightarrow$ 1{colored(X,C):color(C)}1.

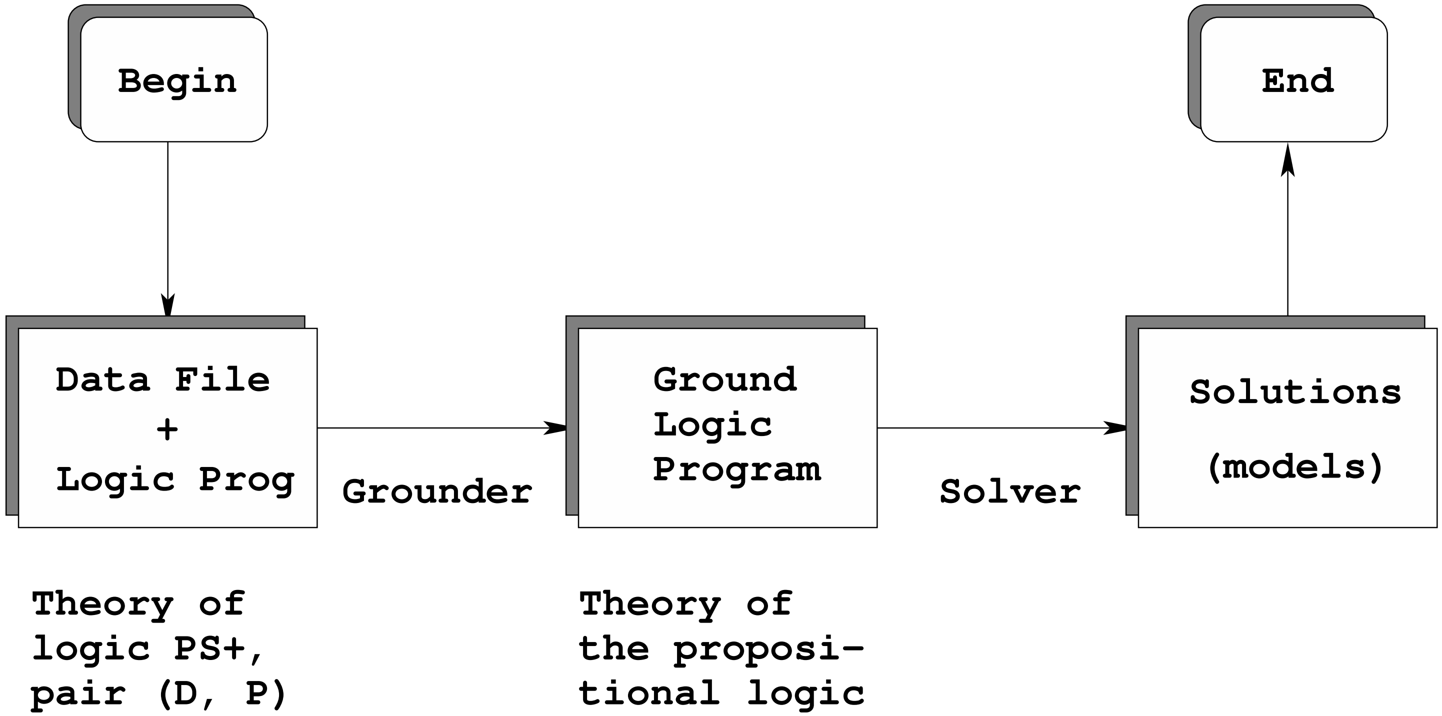
  % No two adjacent vertices have the same color

  edge(X, Y) $\wedge$ colored(X, C) $\wedge$ colored(Y, C) $\rightarrow$ $\perp$.

Computing with Logic PS+

- Existing system for computing models of $PS+$ theories: psgrnd and aspps
  - psgrnd: a grounder that grounds logic $PS+$ theories
  - aspps: a complete solver that computes models of ground logic $PS+$ theories

Flow Chart



Begin
End
Data File
+
Logic Prog
Grounder
Ground
Logic
Program
Solver
Solutions
(models)
Theory of
logic PS+,
pair (D, P)
Theory of
the proposi-
tional logic

Advantages of Logic PS+

- Stable model semantics is more complex to understand
- Logic $PS/PS+$ is close to first-order/propositional logic
- We can use off-the-shelf SAT solvers in addition to solvers, such as aspps, designed for logic $PS/PS+$

Goals for the Presentation

- Provide a broad overview of my research area $\surd$
- Introduce relevant technical background $\surd$
- Describe logic $PS+$ - a focus for my research $\surd$
- **Present my current work**
- Discuss research directions to pursue

Local Search in Logic PS+

- Local search is proved to be effective in solving SAT problems
- We want to experiment with local search in logic PS+
  - Ground logic PS programs are propositional theories $-$ we can use local search SAT solvers directly;
  - Ground logic $PS+$ programs contain c-atoms $-$ we need to modify local search SAT solvers;

Difficulties in Dealing with C-atoms

- Break-count is easy to count in a propositional CNF formula
  - watch only the clauses that have single *true* literal
- It is hard to count when we have c-atoms
  - watching clauses with single *true* literal will not work
  - example 1: $\neg a \vee 1\{a, b\}1$, with $a =$ **f** and $b =$ **f**
  - example 2: $\neg a \vee 1\{a, b\}1$, with $a =$ **f** and $b =$ **t**

Virtual Break-count

- Facts:
  - We cannot apply WalkSAT-Free algorithm directly
  - We can unfold each c-atom into its equivalent in propositional theory
  - We can count WalkSAT-Free break-count w.r.t. the new theory containing unfolded c-atoms
- Bad news: the new theory may become very large
- Good news: we do not need to really unfold each c-atom
- Idea: Break-count w.r.t. the new theory can be computed instead of being counted

Virtual Break-count (contd.)

- Observations:
  - The translation of a c-atom into a propositional theory has a regular form
  - Computing the number of clauses that only have one *true* literal given current truth assignment is a combinatorial problem.
- Break-count can be expressed in terms of $\binom{n}{k}$

Translation of C-atoms

- Two ways of translating each c-atom into a propositional CNF formula
  1. Using a brute-force way that does not introduce extra propositional variables but ends up with a huge propositional theory;
  2. Using extra variables to model c-atoms that does not explode the size of the theory too much but adds too much structured information to the program;

Straightforward Translation

- Upperbound C-atom $\{a_1, \ldots, a_k\}n$
  - $\neg a_{i_1} \vee ... \vee \neg a_{i_{n+1}}$, for any $n+1$ atoms $a_{i_1}, ..., a_{i_{n+1}}$
  - Intuition: given any subset $\{a_{i_1}, ..., a_{i_{n+1}}\}$ of size $n+1$, there is at least one atom that has the truth value **f**.
- Lowerbound C-atom $m\{a_1, \ldots, a_k\}$
  - translation is similar to that of the upperbound case

Approximating Virtual Break-count

- $\binom{n}{k}$ may become large and beyond the storing capability of computers
- Use of some software packages, such as GNU Calc that operates arbitrary precision integers, makes the computation much slower
- Use of floating-point arithmetic also makes the computation slow
- Use of approximations to compute large integer numbers is better
- How to approximate virtual break-count is still an open topic for research

Approximations

- We have tried several approximations of computing $\binom{n}{k}$:
  1. Stirling approximation:
     - uses float-point arithmetic;
  2. Linear approximation:
     $\binom{n}{k} = n \times k$ if $k \leq n/2$; $\binom{n}{k} = n \times (n-k)$ otherwise;
     - results are accurate when $k = 0, 1$
  3. Quadratic approximation:
     $\binom{n}{k} = a \times k^2 + b \times k + 1$, where $a = (n^2 - 5n + 2)/4$ and $b = (-n^2 + 9n - 6)/4$.
     - results are accurate when $k = 0, 1, 2$

Exploiting the Structure of the Program

- Logic $PS+$ programs can often be divided into two parts:
  - a part containing c-atoms
  - a part consisting of only propositional clauses
- Virtual break-count computation can be avoided when
  - we can easily satisfy the part that has c-atoms
  - we can keep the part satisfied during flipping
- Specialized flip techniques include (not exhaustively)
  - double flip
  - permutation flip
- They depend on the structure of the theory

Constraints Suitable for Double Flip

- Example: at most/at least/exactly n atoms should be *true*

$$\{in\_cover(X) : vertex(X)\}n.$$

- Condition: Grounding the rule will yield a set of unit disjoint c-atom rules
- Intuition:
  - It is easy to generate an initial assignment to satisfy all those rules
  - It is easy to maintain the truth values of all those rules by allowing more than one flip

Double Flip Algorithm

$Flip(F, \sigma, a)$

Input:

- F - a logic $PS+$ formula
- σ- current truth assignment
- a - the chosen atom to flip

Output:

- updated (F, σ) after a is flipped

Double Flip Algorithm Contd.

```
BEGIN
1.  If a occurs in one of the disjoint unit
c-atoms and flipping a will break it, then
2.      pick the best opposite atom in that
c-atom w.r.t.  break-count;
3.      flip the chosen atom;
4.  End if
5.  Flip a;
6.  Update (F,σ);
7.  return (F,σ);
END
```

Variants of Double Flip

- Fact: we need to choose two atoms in performing double flip
- Possible changes to the $Pick$-$Atom$ algorithm
  - follow the original $Pick$-$Atom$ algorithm
  - choose a such that $BC(a) + BC(b)$ is minimized, where b is the best opposite atom w.r.t. a;
- Possible changes to the $Flip$ algorithm
  - allow to break unit c-atom rules during random walk step

Experiments and Results

- In all of the following experiments, we used machines with P4 1.5GHz CPU, 1.0GB memory, running Linux version 2.4.18 (gcc version 2.95.3)
- We considered the following NP-complete problems: vertex cover, dominating set problem, 4-colorability problem, Schur number (5), open Latin square problem and n-queens problem.
- In local search algorithms, we specified the followings:
  - Number of Retry: 10
  - Number of Flips in Each Try: 100000

Format of Measurements

- Results are shown in the following format: $t/f/s/r$, where
  - t is the running time of all tries and all runs;
  - f is the number of flips in all success tries of all runs that find at least one solution;
  - s is the success rate in all runs that find at least one solution;
  - r is the ratio of the number of runs that find solution over the total number of runs.

Vertex Cover

- Noise: 0.1 (10 : 100)
- Graph size: 2000 vertices, 4000 edges
- Number of Graph's: 50 randomly generated

Instance	VBC2	VBC3	DBF
1035 (8 / 50)	137/34792/63%/75%	133/32072/78%/62%	310/51283/82%/87%
1040 (24 / 50)	110/33638/87%/83%	110/34101/84%/83%	220/36392/87%/95%
1045 (35 / 50)	77/22739/87%/91%	79/23281/89%/88%	152/23481/85%/100%
1050 (50 / 50)	41/21063/90%/96%	44/21531/90%/96%	48/24551/95%/100%
1055 (50 / 50)	18/13202/99%/100%	17/13026/100%/100%	16/10907/100%/100%

4-Colorability

- Noise: 0.1 (10 : 100)
- Number of Graph's: 50 randomly generated

Instance	DBF+SW	ZCHAFF	WSAT
1000X2000 (50/50)	0/1267/99%/100%	0/0/100%/100%	0/2170/100%/100%
1000X3000 (50/50)	0/5360/99%/100%	0/0/100%/100%	0/10017/100%/100%
1000X3400 (50/50)	1/17429/98%/100%	0/0/100%/100%	1/42280/100%/100%
1000X3800 (50/50)	15/136968/40%/96%	21/0/100%/100%	6/154107/17%/74%
1000X3850 (50/50)	17/145976/23%/70%	110/0/100%/100%	7/159637/12%/34%
1000X3860 (50/50)	18/150745/18%/57%	225/0/100%/100%	7/155154/13%/26%
1000X3870 (50/50)	18/152365/13%/66%	368/0/100%/100%	7/153467/10%/20%
1000X3880 (50/50)	18/149555/16%/34%	438/0/100%/100%	7/155585/10%/16%
1000X3890 (50/50)	19/159549/15%/24%	891/0/100%/100%	7/183606/12%/16%
1000X3900 (50/50)	19/140410/12%/34%	1137/0/100%/100%	7/155648/12%/8%

Goals for the Presentation

- Provide a broad overview of my research area $\surd$
- Introduce relevant technical background $\surd$
- Describe logic $PS+$ - a focus for my research $\surd$
- Present my current work $\surd$
- **Discuss research directions to pursue**

Research Directions - my Future Work

- Extend logic $PS+$
- Improve solvers to compute models of logic $PS+$ programs
- Demonstrate applicability in knowledge reasoning (KR)
- Demonstrate applicability in solving hard combinatorial problems
- Implement hybrid programming systems

Extending Logic PS+

- Additional high-level constraints (such as weight constraints)
- Aggregates: “max”, “sum”, “avg”, . . .
- Ability to specify optimization tasks, such as to find minimum weight vertex cover or maximum average weight vertex cover

Improvement on Solvers

- Local search solvers:
  - design an automated way to use solutions of a smaller instance of the problem and extend them to solutions of a larger instance;
  - improve local search in the structured programs;
- Complete solvers: new heuristics; learning components; backjumping schema; restart policy; efficient data structure
- Both:
  - integrating grounding into solvers to avoid full grounding
  - integrating systematic search into local search or vice versa

Applicability in KR

- Develop $PS+$ theories for diagnosis, abduction and planning
- Develop or use specialized front-ends such as action language A, planning language STRIP
  - Existing front-end: puzzle language $ConstraintLingo$ (Raphael Finkel)
- Represent stable logic programming in $PS+$ (similar to cmodels and as-sat)

Applicability in Solving Hard Problems

- Computing Ramsey-type numbers (at least, bounds)
  - Ramsey numbers
  - Schur numbers
  - Vander Werden numbers

Hybrid Systems

- Integrating PS+ programming environment with external program libraries
- Implementing PS+ APIs for other programming environments

End of the Presentation

Thank you!

Logic Programming with Negation

- **Logic programs**: collection of **rules** of the following form:
  $p \leftarrow b_1, \ldots, b_m, not\ c_1, \ldots, not\ c_n.$
- First-order logic interpretation: rules as implications and not as negations
- Intended (procedural) reading of rules: if all of $b_1, \ldots, b_m, not\ c_1, \ldots, not\ c_n$ are computed, we can compute p

Logic Programming with Negation (contd.)

- More appropriate semantics:
  - stable model semantics (Gelfond and Lifschitz)
  - supported model semantics (Clark's completion)
  - well-founded model semantics ()

Logic Programming as ASP System

- Existing software to compute stable models of a logic program: Smodels, DLV
- Procedures to solve a problem:
  - encode the problem as a logic program P so that stable models of P represent solutions
  - use Smodels (or dlv) to find stable models of P
  - reconstruct solutions from the stable models

Example - 3 Colorability Problem

- Instance: An undirected graph and 3 colors
- Question: Can we color the graph such that
  1. each vertex gets exactly one color;
  2. no two adjacent vertices have the same color;
- Example Graph: A rectangle with 4 vertices and 4 edges

Smodels Encoding for 3-Col

```
% Define colors and the graph
color(red).  color(blue).  color(green).
vtx(1).  vtx(2).  vtx(3).  vtx(4).
edge(1,2).  edge(2,3).  edge(3,4).  edge(4,1).
% Each vertex get exactly one color
colored(X,red)←vtx(X),not colored(X,blue),not
colored(X,green).
colored(X,green)← vtx(X),not colored(X,red),not
colored(X,blue).
colored(X,blue)← vtx(X),not colored(X,green),not
colored(X,red).
% No two adjacent vertices have the same color
f← vtx(X),vtx(Y),edge(X,Y),color(C),
colored(X,C),colored(Y,C),not f.
```

Default Logic

- A generalization of logic programming with negation
- Logic program rules are replaced by more general inference rules: **defaults**
  - syntax: $\frac{\alpha : M\beta_1, \ldots, M\beta_n}{\gamma}$
  - rule $p \leftarrow b_1, \ldots, b_m, not\, c_1, \ldots, not\, c_n$ is interpreted as default $\frac{b_1 \wedge \cdots \wedge b_m : M\neg c_1, \ldots, M\neg c_n}{p}$

Some Notions

- $C = l_1 \vee l_2 \vee ... \vee l_r \vee c_1 \vee c_2 \vee ... \vee c_p$
- $C_h = \bigwedge_{i=1}^{\binom{k_h}{n_h+1}} P_i \wedge \bigwedge_{j=1}^{\binom{k_h}{k_h-m_h+1}} N_j$ for c_h
- $C = l_1 \vee l_2 \vee ... \vee l_r \vee C_1 \vee C_2 \vee ... \vee C_p$

Three Cases in Flipping an Atom

- Case 1: after flipping an atom $a \in C$, at least one normal literal is *true*;
- Case 2: after flipping $a \in C$ from *true* to *false*, all normal literals are *false*;
- Case 3: after flipping $a \in C$ from *false* to *true*, all normal literals are *false*;

Case 1

- A trivial case because clause C will not contribute to the break-count of a

Case 2

- Virtual break-count of a is updated by the following formula:

$$VBC(a) \leftarrow BC(a) + \prod_{i=0}^{p} V_i(a) - \prod_{i=0}^{p} U_i(a)$$

where $V_i(a) = \begin{cases} \binom{neg_i}{k_i - m_i} + \binom{neg_i}{k_i - m_i + 1} + \binom{pos_i - 1}{n_i + 1}, & \text{if } a \in c_i \\ \binom{neg_i}{k_i - m_i + 1} + \binom{pos_i}{n_i + 1} & \text{, otherwise} \end{cases}$

and $U_i(a) = \begin{cases} \binom{neg_i}{k_i - m_i + 1} + \binom{pos_i - 1}{n_i + 1}, & \text{if } a \in c_i \\ \binom{neg_i}{k_i - m_i + 1} + \binom{pos_i}{n_i + 1} & \text{, otherwise} \end{cases}$

Case 3

- Virtual break-count of a is updated by the following formula:

$$VBC(a) \leftarrow BC(a) + \prod_{i=0}^{p} \overline{V_i}(a) - \prod_{i=0}^{p} \overline{U_i}(a)$$

where $\overline{V}_i(a) = \begin{cases} \binom{pos_i}{n_i} + \binom{pos_i}{n_i+1} + \binom{neg_i-1}{k_i-m_i+1}, \text{ if } a \in c_i \\ \binom{pos_i}{n_i+1} + \binom{neg_i}{k_i-m_i+1}, \text{ otherwise} \end{cases}$

and $\overline{U}_i(a) = \begin{cases} \binom{pos_i}{n_i+1} + \binom{neg_i-1}{k_i-m_i+1}, \text{ if } a \in c_i \\ \binom{pos_i}{n_i+1} + \binom{neg_i}{k_i-m_i+1} \quad, \text{ otherwise} \end{cases}$

Example

- Let $C = \neg a \vee 1\{a, b\}1$. and current truth assignment $I_0 = \{b\}$
- $1\{a, b\}1$ is equivalent to the following two propositional clauses:
  $\neg a \vee \neg b$ and $a \vee b$
- C is equivalent to
  $C_1' = \neg a \vee \neg a \vee \neg b$ and $C_2' = \neg a \vee a \vee b$
- flipping atom a will breaking C_1' and will not break C_2'
- thus the break-count of atom a is 1.

Example Contd.

- $k = 2, m = 1, n = 1, p = 1, pos = 1, neg = 1$ and $a \in 1\{a, b\}1$,
- $\overline{V}_1(a) = \binom{pos}{n} + \binom{pos}{n+1} + \binom{neg-1}{k-m+1} = \binom{1}{1} + \binom{1}{2} + \binom{0}{2} = 1$
- $\overline{U}_1(a) = \binom{pos}{n+1} + \binom{neg-1}{k-m+1} = \binom{1}{2} + \binom{0}{2} = 0$
- virtual break-count $\overline{V}_1(a) - \overline{U}_1(a) = 1$,

Generalized Double Flip

- We can apply double flip idea in SAT solvers
  - Idea: select an arbitrary set of disjoint rules (not necessarily to be the unit c-atom rules) and perform double flip on them
- We need to solve one problem:
  - how to choose the set of disjoint rules

Grounding of Logic PS Formulas

- Grounding of an e-atom $p(t)$:
  - t is a tuple that contains '_';
  - let t' be a tuple obtained by replacing all occurrences of '_' with constants in the Herbrand universe;
  - grounding of $p(t)$ is defined as $p^d(t) = p(t')$;
- Grounding of a rule r:
  - let r^d be the rule obtained by grounding all e-atoms in r;
  - instance of r is defined as $r^d\theta$, where θ is a ground substitution;
  - grounding of r is defined as $ground(r) = r^d\theta : \forall\theta$
- Grounding of a logic PS program T is defined as $ground(T) = \bigcup_{r \in T} ground(r)$