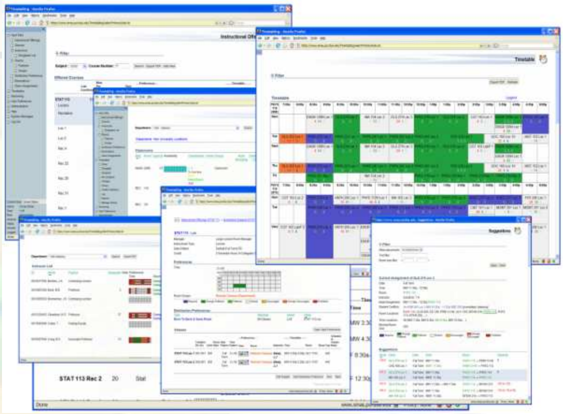
ITC2007 Solver Description: A Hybrid Approach

PATAT 2008

Tomáš Müller

Objectives



- Use the Constraint Solver Library
  - Open source (GNU LGPL)
  - Local-search based framework
  - Written in Java
  - Used in our university timetabling system UniTime
    - Course Timetabling, Examination Timetabling, Student Sectioning
    - Applied in practice on a large university-wide problem (at Purdue University)
    - Web based, open source (GNU GPL), also written in Java
    - More information available at http://www.unitime.org
      - Including download, documentation, demo, real-life benchmarks, …
- Apply the same algorithm for all three tracks
  - With only minimal changes to reflect different problem formulations
    - Problem model, neighborhoods
- Compare used techniques and achieved results with other competitors
  - Further improve the constraint solver library

Constraint Solver Library

- Constraint model
  - Variable, Value, Constraint, Model, Neighborhoods, etc.
    - Abstract classes or interfaces
      - E.g., a lecture can be modeled as a variable, time & room assignment as a value
    - Including relations between these primitives
      - Variable has a domain, constraint works with a set of variables, etc.

- Local-search based, however
  - Operates over feasible, though not necessarily complete, solutions
  - Feasibility is assured automatically
    - Via notifications that are send between variables and their constraints
    - Constraints can maintain information to ensure quick feasibility checks
      - E.g., each room can have a table containing current assignments

        f : time slot → a lecture or empty

Constraint Solver Library

- Default search algorithm and strategies
  - Iterative forward search
    - Guided by neighborhood selection, termination, and solution comparison heuristics

```
while (termination.canContinue(solution)) {
  Neighbour n = neighbourSelection.select(solution);
  if (n!=null) n.assign(solution);
  if (solutionComparator.isBetterThanBest(solution)) solution.saveBest();
}
```

  - Conflict-based statistics
    - If $A=a$ is unassigned because of the $B=c$
      - A counter $CBS[A\neq a, B=c]$ is incremented
    - Conflicts are weighted by their past occurrences

$$A \neq a \Leftarrow \begin{cases} 3 \times B = a \\ 4 \times B = c \\ 2 \times C = a \\ 120 \times D = a \end{cases}$$

  - Minimal Perturbation Problem
    - Original solution, modified problem
      → adopted solution should differ as little as possible
  - Extendable

Competition Tracks

- Track 1: Examination Timetabling
  - Exams, students, periods, rooms
    - Two or more exams can be in one room.
    - No direct student conflicts, period lengths, room capacities
    - Additional constraints (precedence, room exclusivity, same/different period)
  - Penalizations for
    - Two exams in a row or in a day, period spread (two exams closer than given number of periods)
    - Room and period penalties, mixed durations, large exams in later periods

Competition Tracks

- Track 2: Post Enrollment Course Timetabling
  - Events, students, time slots (5 days, each with 9 slots), rooms
    - No direct student conflicts, room capacities & features
    - Extension of International Timetabling Competition from 2003
      - Added event availability, precedence constraints
  - Penalizations for
    - Last slot a day, more than two events consecutively, single event a day

Competition Tracks

- Track 3: Curriculum-based Course Timetabling
  - Lectures, courses, curricula, periods, rooms, teachers
    - Lectures organized into courses, availability, minimal number of days
    - Courses grouped into curricula
    - Lectures of the same curricula or teacher must be assigned in different periods
  - Penalizations for
    - Room capacity (room size < number of student in a course)
    - Spread of lectures of a course into minimal number of days
    - Curriculum compactness (a lecture not adjacent to another lecture of the same curricula)
    - Room stability (lectures of the same course in different rooms)

Constraint Solver Library Example

```
public class TimetablingModel extends Model {
  public Vector<Variable> variables(); // set of events
  public Vector<Constraint> constraints(); // rooms, students, precedences
    // total score (sum of Student.score() over all students)
  public int getTotalValue();
}
public class Event extends Variable {
  public Set<Student> students();
  public Set<Room> rooms();
  public boolean isAvailable(int slot);
  public Set<Placement> values() {
    Set values<Placement> = new Set();
    for (int time=0;time<45;time++) {
      if (!isAvailable(time)) continue;
      for (Room room : rooms())
        values.addElement(new Placement(this, time, room));
    }
    return values;
  }
}
public class Placement extends Value {
  public Event variable();
  public int time();
  public Room room();
  public int toInt(); //change in score if this assigned
}
```

Constraint Solver Library Example

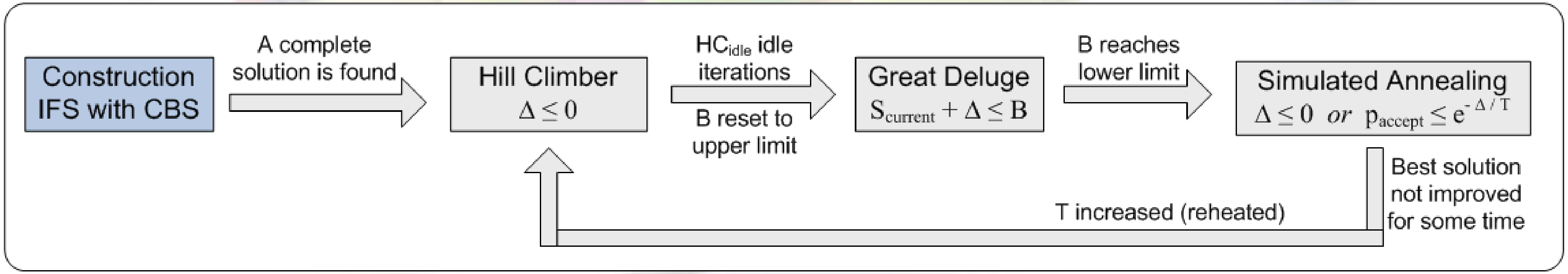
```
public class Student extends Constraint {
  private Placement[] iTable = new Placement[45];
  public void assigned(long iteration, Placement value) {
    super.assigned(iteration, value);
    iTable[value.time()]=value;
  }
  public void unassigned(long iteration, Placement value) {
    super.unassigned(iteration, value);
    iTable[value.time()]=null;
  }
  public void computeConflicts(Placement value, Set conflicts) {
    if (iTable[value.time()]!=null) conflicts.add(iTable[value.time()]);
  }
  public int score() {
    int score = 0;
    for (int d=0;d<5;d++) { int inRow = 0, eventsADay = 0;
      for (int t=0;t<9;t++) { int slot = d*9 + t;
        if (iTable[slot]!=null) { inRow++; eventsADay++; if (t==8) score++; }
        else inRow = 0;
        if (inRow>2) score++;
      }
      if (eventsADay==1) score++;
    }
    return score;
  }
}
```

Constraint Solver Library Example

```
public class Precedence extends BinaryConstraint {
  public void computeConflicts(Placement value, Set conflicts) {
    if (first().equals(value.variable())) {
      Placement second = second().assignment();
      if (second!=null && value.time()>=second.time()) conflicts.add(second);
    } else {
      Placement first = first().assignment();
      if (first!=null && first.time()>=value.time()) conflicts.add(first);
    }
  }
}
public class Room extends Constraint {
  private Placement[] iTable = new Placement[45];
  public void assigned(long iteration, Placement value) {
    super.assigned(iteration, value);
    if (this.equals(value.room())) iTable[value.time()]=value;
  }
  public void unassigned(long iteration, Placement value) {
    super.unassigned(iteration, value);
    if (this.equals(value.room())) iTable[value.time()]=null;
  }
  public void computeConflicts(Placement value, Set conflicts) {
    if (this.equals(value.room()) && iTable[value.time()]!=null)
      conflicts.add(iTable[value.time()]);
  }
}
```

Competition Solver

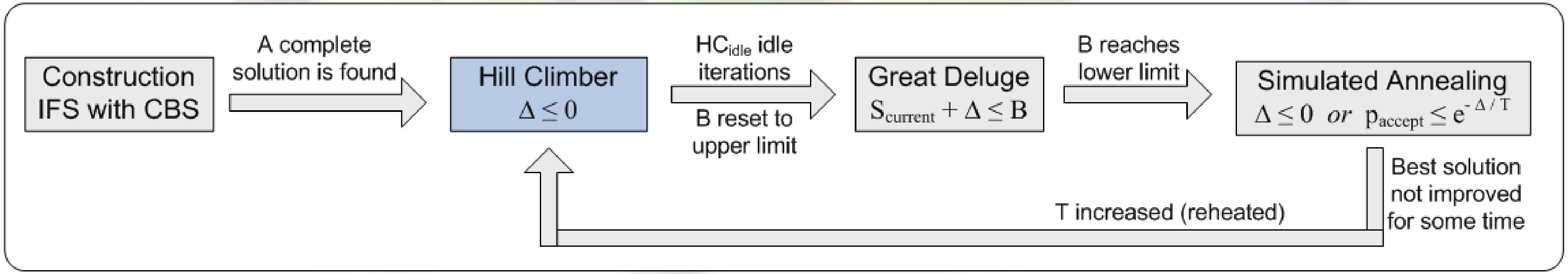
- 1. Construction phase
  - Iterative forward search with conflict-based statistics
  - Starts with all variables unassigned
  - In each iteration:
    - Select the “hardest” unassigned variable (domain size / # hard constraints)
    - A best value is selected
      - Change in objective function
      - Hard conflicts weighted by conflict-based statistics
    - Value is assigned, conflicting variables are unassigned
  - Until a complete solution is found



Competition Solver

- 2. Hill climber
  - In each iteration:
    - Generate a move
      - Random selection of one of given problem-specific neighborhoods
      - Random generation of a neighbor move (E.g., moving a selected class into a different room)
      - Only not conflicting neighbors are considered
    - A move is accepted when it does not worsen the overall solution value
  - Until a given number of idle (not improving) iterations

Number of Idle Iterations HC_{idle} = 25,000 (1); 50,000 (2&3)



Constraint Solver Library Example

```
public class HillClimber implements NeighbourSelection {

  private Vector<NeighbourSelection> iNeighborhoods; //list of neighborhoods
  private int iIdle = 0; //number of idle iterations

  public Neighbour select(Solution solution) {
    while (iIdle<25000) {
      NeighbourSelection neighbour = ToolBox.random(iNeighborhoods);
      Neighbour n = neighbour.select(solution);
      iIdle++;
      if (n==null) continue;
      if (n.value()<0.0) { iIdle = 0; return n; }
      else if (n.value()==0) return n;
    }
    return null;
  }

}
```

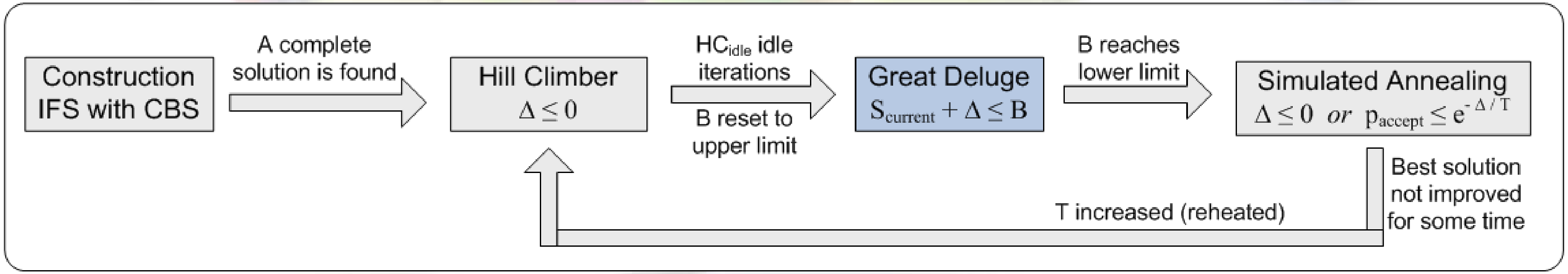
Competition Solver

- 3. Great Deluge
  - Bound
    - Initialized to $B = GD_{ub} \cdot S_{best}$
  - In each iteration:
    - Generate a move
      - Same as in Hill Climber
    - A move is accepted when the new solution value does not exceed the bound
    - Bound is decreased after every iteration $B = B \cdot GD_{cr}$
  - Repeated until bound reaches lower limit $GD_{lb}^{at} \cdot S_{best}$
    - Reheat: $B = GD_{ub}^{at} \cdot S_{best}$
      - Where *at* is the number of reheats without best found solution being improved

Upper Bound GD_{ub} = 1.12 (1); 1.10 (2); 1.15 (3)

Cooling Rate GD_{cr} = 0.99999988 (1); 0.9999998 (2); 0.99999986 (3)

Lower Bound GD_{lb} = 0.9



Competition Solver

- 4. Simulated Annealing
  - Temperature
  - In each iteration:
    - Generate a move
      - Same as in Hill Climber
    - A move is accepted if it is not worsening or with probability $e^{-\Delta / T}$
    - After each $SA_{cc} \cdot TL$ iterations, temperature decreased by a cooling rate $T = T \cdot SA_{cr}$
  - Repeated until $SA_{rc} \cdot SA_{cc} \cdot TL$ of idle (not improving) iterations is reached
    - Temperature reheated $T = T \cdot SA_{cr}^{-1.7 \cdot SA_{rc}}$

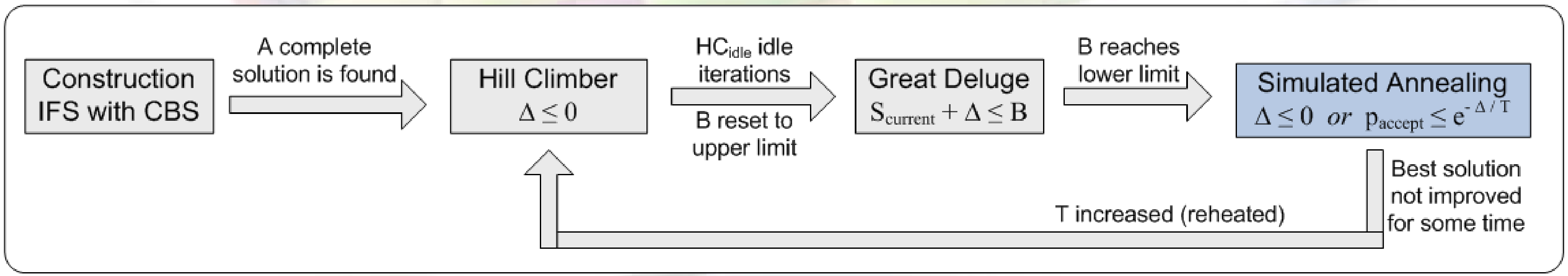
Initial Temperature SA_{it} = 1.5 (2); 2.5 (3)

Cooling Coeficient SA_{cc} = 5 (2); 7 (3)

Temperature Length TL ~ sum of domain sizes

Cooling Rate SA_{cr} = 0.97 (2); 0.82 (3)

Reheat Coeficient SA_{rc} = 7 (2); 7 (3)



Neighborhoods

- Track 1: Examination Timetabling
  - Exam Swap
    - select exam, new period and room, assign or swap with conflicting exam
    - try following periods and rooms if two or more conflicts or unable to swap
  - Period Change, Room Change, Period and Room Change
    - select exam, new period/room, assign when no conflict
    - otherwise try following periods/rooms
  - Period Swap, Room Swap
    - select exam, new period/room, if one conflicting exam swap exams
    - otherwise try following periods/rooms

  - Comments
    - Simulated annealing step not used (after great deluge phase, it gets back to great deluge phase, but with new bound)

Neighborhoods

- Track 2: Post Enrollment Course Timetabling
  - Time Move, Room Move
    - select event, new time slot/room, assign when no conflict
  - Event Move
    - select event, new time slot and room, assign when no conflict try to swap when one conflict
  - Event Swap
    - select two events, try to swap times (can pick different rooms)
  - Precedence Swap

    Selected less often than the others
    - select violated precedence constraint, try to reassign one event (select different time slot and room) so that the constraint is satisfied
  - Comments
    - Soft constraints are ignored during construction phase
    - It is allowed to assign an event into a time with no room or to violate precedence constraint

Violations weighted by one at the beginning, increased by one every 1,000 iterations

Neighborhoods

- Track 3: Curriculum-based Course Timetabling
  - Time Move, Room Move
    - select lecture, new time slot/room, assign when no conflict
  - Lecture Move
    - select lecture, new time slot and room, assign when no conflict try to swap when one conflict
  - Room Stability Move
    - select course, room, try assign all lectures in the rooms, may swap lectures between rooms
  - Min Working Days Move
    - select course, select a day with two or more lectures, try to move a lecture to another (unused) day
  - Curriculum Compactness Move
    - select course and not adjacent lecture, try move lecture to some adjacent time

Selected less often than the others

Constraint Solver Library Example

```
public class RoomMove implements NeighbourSelection{

  public Neighbour select(Solution solution) {
    // select an event at random
    Event event = ToolBox.random(solution.model().variables());
    // keep time
    int time = event.assignment().time();
    // select a room at random (from the rooms where the event can take place)
    int rx = ToolBox.random(event.rooms().size());
    // iterate rooms starting from rx, look for the first available one
    for (int r=0;r<event.rooms().size();r++) {
      Room room = event.rooms().get((r+rx)%event.rooms().size());
      // skip currently assigned room
      if (room.equals(event.assignment().room())) continue;
      Placement placement = new Placement(event,time,room);
      if (solution.model().computeConflicts(placement).isEmpty())
        return new SimpleNeighbour(event,placement); //reassignment of event
    }
    return null; // no room available
  }

}
```

Results of Track 1: Examination Timetabling

Submitted results (best solution of 100 runs)

Instance Number	1	2	3	4	5	6	7	8
Two Exams in a Row	42	0	1275	7533	40	3700	0	0
Two Exams in a Day	0	10	2070	3245	0	0	0	0
Period Spread	2534	0	5193	3958	1361	19900	3628	6718
Mixed Durations	100	0	0	0	0	75	0	0
Larger Exams Constraints	260	380	840	105	1440	375	460	380
Room Penalty	1150	0	0	0	0	1250	0	125
Period Penalty	270	0	190	1750	100	475	0	342
Overall Value	**4356**	**390**	**9568**	**16591**	**2941**	**25775**	**4088**	**7565**

Final ordering (best solutions run by organizers)

Instance Number	1	2	3	4	5	6	7	8	9	10	11	12	rank
T. Müller	**4370**	**400**	**10049**	**18141**	**2988**	**26585**	**4213**	**7742**	**1030**	16682	**34129**	5535	13.3
C. Gogos	5905	1008	13771	18674	4139	27640	6572	10521	1159	-	43888	-	23.4
M. Atsuta et al.	8006	3470	17669	22559	4638	29155	10473	14317	1737	15085	-	**5264**	28.4
G. Smet	6670	623	-	-	3847	27815	5420	-	1288	**14778**	-	-	28.6
N. Pillay	12035	2886	15917	23582	6860	32250	17666	15592	2055	17724	40535	6310	33.8

UNI|time.org

Results of Track 2: Post Enrollment Course Timetbl.

Submitted results (best solution of 100 runs)

Instance Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Distance to Feasibility	0	0	0	0	0	0	0	0	0	**57**	0	0	0	0	0	0
More than Two in a Row	728	1093	73	111	0	8	2	0	881	1268	118	169	70	2	0	2
One Class on a Day	23	21	132	283	0	0	3	0	16	33	177	233	1	0	0	4
Last Time Slot of a Day	579	1040	0	0	0	5	0	0	998	1139	52	51	3	0	0	0
Overall Value	**1330**	**2154**	**205**	**394**	**0**	**13**	**5**	**0**	**1895**	**2440**	**347**	**453**	**74**	**2**	**0**	**6**

Final ordering (best solutions run by organizers)

Instance Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	rank
H. Cambazard et al.	571	993	**164**	310	**5**	**0**	6	**0**	1560	2163	**178**	**146**	**0**	1	**0**	2	**0**	**0**	1824	445	**0**	29	**238**	**21**	13.9
M. Atsuta et al.	61	547	382	529	**5**	**0**	**0**	**0**	**0**	**0**	548	869	**0**	**0**	379	191	1	**0**	-	1215	**0**	**0**	430	720	24.4
M. Chiarandini et al.	1482	1635	288	385	559	851	10	**0**	1947	1741	240	475	675	864	**0**	**1**	5	3	1868	596	602	1364	688	822	28.3
C. Nothegger et al.	**15**	**0**	391	**239**	34	87	**0**	4	**0**	**0**	547	32	166	**0**	**0**	41	68	26	**22**	-	33	**0**	-	30	29.5
T. Müller	1861	-	272	425	8	28	13	6	-	-	263	804	285	110	5	132	72	70	-	878	40	889	436	372	31.3

Results of Track 3: Curriculum Course Timetabling

Submitted results (best solution of 100 runs)

Instance Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Room Capacity	4	0	0	0	0	0	0	0	0	0	0	0	0	0
Minimum Working Days	0	15	10	5	180	15	0	5	35	5	0	255	10	5
Curriculum Compactness	0	28	62	30	114	26	14	34	68	4	0	76	56	48
Room Stability	1	0	0	0	4	0	0	0	0	0	0	0	0	0
Overall Value	5	43	72	35	298	41	14	39	103	9	0	331	66	53

Final ordering (best solutions run by organizers)

Instance Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	rank
T. Müller	**5**	51	84	37	330	**48**	**20**	41	109	**16**	**0**	**333**	**66**	59	84	**34**	**83**	83	**62**	**27**	**103**	12.9
Z. Lu et al.	**5**	55	**71**	43	**309**	53	28	49	**105**	21	**0**	343	73	**57**	**71**	39	91	69	65	47	106	16.7
M. Atsuta et al.	**5**	**50**	82	**35**	312	69	42	**40**	110	27	**0**	351	68	59	82	40	102	**68**	75	61	123	17.6
M Geiger	**5**	111	128	72	410	100	57	77	150	71	**0**	442	622	90	128	81	124	116	107	88	174	38.2
M. Clark et al.	10	111	119	72	426	130	110	83	139	85	3	408	113	84	119	84	152	110	111	144	169	42.2

Conclusions

- Success!
  - Winner of two tracks, finalist of all three
  - With a single (hybrid) approach
- Further work
  - More in depth comparison with competition solvers
  - Improvement of the existing solver for university timetabling application
- Applications
  - Examination timetabling at Purdue and Widener Universities
    - Different model, same solver
      - E.g., an exam can be split into multiple rooms if needed, direct student conflicts are allowed (but minimized)
- Additional Information (including source code)
  - http://www.unitime.org/itc2007