SAT Solver dengan DPLL dalam Pemrograman Deklaratif

Taufiq Hidayat[1], Agung Bahariyanto Irhasni[2]
Jurusan Informatika, Fakultas Teknologi Industri
Universitas Islam Indonesia
Yogyakarta
[1]taufiq.hidayat@uii.ac.id, [2]13523204@students.uii.ac.id

***Abstrak*—SAT Solver adalah perangkat lunak untuk menyelesaikan SAT Problem. Penelitian ini bertujuan membangun SAT Solver dengan konsep pemrograman deklaratif dan bahasa pemrograman Prolog dengan menggunakan Algoritma DPLL. Penelitian ini merupakan bagian dari penelitian tentang sistem eksplorasi Formal Context dengan *constraint*. Setelah pengujian, dapat disimpulkan bahwa SAT Solver dari penelitian ini dapat menyelesaikan SAT Problem. Salah satu problem yang diujikan adalah problem Sudoku, yang dinyatakan dalam SAT Problem dengan 729 variabel dan paling sedikit 8829 klausa.**

Kata kunci—SAT Solver, DPLL, pemrograman deklaratif.

I. PENDAHULUAN

SAT Solver adalah perangkat lunak yang digunakan untuk menyelesaikan SAT Problem (*satisfiability problem*), yaitu untuk menentukan apakah sebuah formula logika proposisi itu *satisfiable* atau *unsatisfiable*. Formula logika proposisi yang diselesaikan dengan SAT Solver harus dalam bentuk CNF (*conjunctive normal form*)[1].

Dalam penelitian terdahulu, SAT Solver digunakan untuk menyelesaikan eksplorasi atribut Formal Context dengan batasan (*constraint*)[2]. Salah satu contoh eksplorasi atribut Formal Context dengan *constraint* adalah eksplorasi atribut Many-Valued Context[3], sedangkan salah satu contoh aplikasi dari eksplorasi atribut Formal Context dengan *constraint* adalah untuk menggali aturan asosiasi halaman-halaman web yang dikunjungi seorang pengunjung. Pada kasus terakhir, Formal Context adalah halaman-halaman web yang dikunjungi setiap pengunjung dalam satu sesi kunjungan. *Constraint* pada kasus ini adalah struktur web[4].

Bagian eksplorasi attribute Formal Context dengan batasan yang diselesaikan dengan SAT Solver adalah *entailment problem*. Langkah penyelesaian problem entailment dengan SAT Solver adalah sebagai berikut:

1. Problem entailment dinyatakan sebagai SAT Problem.
2. SAT Problem disimpan pada sebuah file teks.
3. SAT Solver dijalankan dengan input berupa file teks untuk *entailment problem*.

Dengan langkah ini, yang menjadi permasalahan adalah proses untuk menyelesaikan *entailment problem* harus dengan input dan output melalui file teks. Sudah diketahui bahwa pengakesan file membutuhkan waktu yang signifikan. Padahal dalam menyelesaikan satu kasus, sistem eksplorasi Formal Context ini harus menyelesaikan banyak sekali *entailment problem* sehingga hal ini mempengaruhi unjuk kerja sistem tersebut.

Untuk menghindari pengaksesan file ini, diperlukan sebuah SAT Solver yang terintegrasi dengan sistem eksplorasi Formal Context tersebut. Diharapkan dengan SAT Solver yang terintegrasi akan dapat meningkatkan unjuk kerja sistem eksplorasi Formal Context tersebut. SAT Solver ini harus dibuat dalam bahasa pemrograman deklaratif, yaitu Prolog, karena sistem ekplorasi Formal Context dibuat dalam bahasa tersebut.

Ada beberapa SAT Solver yang sudah dibangun dengan menggunakan Prolog. Di antaranya adalah A Pearl[5], yang menggunakan metode *watched literal*. SAT Solver yang lain melakukan integrasi MINISAT dalam C++ dengan Prolog[6]. Namun, belum ada SAT Solver dalam Prolog yang menggunakan Algoritma DPLL[7].

Algoritma DPLL dan pengembangannya adalah algoritma yang banyak dipakai oleh SAT Solver terbaik pada kompetisi SAT/*SAT Competition* (http://satcompetition.org). Bisa dikatakan bahwa Algoritma DPLL adalah algoritma dengan unjuk kerja terbaik. Salah satu pengembangan dari Algoritma DPLL ini adalah *Conflict-Driven Clause Learning*[8]. SAT Solver yang menjadi juara kompetisi 2017 menggunakan Algoritma DPLL dengan CDCL ini.

Oleh karena itu, pada penelitian ini SAT Solver dengan Prolog dibangun dengan menggunakan Algoritma DPLL. Selain diharapkan dapat meningkatkan sistem eksplorasi Formal Context, SAT Solver ini diharapkan juga dapat menjadi dasar untuk membangun SAT Solver dalam pemrograman deklaratif.

II. LANDASAN TEORI

A. Formula Proposisi dan CNF

Formula proposisi adalah formula dalam bentuk logika proposisi. Logika proposisi adalah logika yang disusun dari kalimat proposisi, yaitu kalimat yang dapat ditentukan nilai kebenarannya. Sebuah proposisi dinyatakan dalam sebuah variabel proposisi yang memiliki 2 kemungkinan nilai, benar (*true*) atau salah (*false*)[9].

[*]Penelitian ini disponsori oleh Jurusan Informatika, Fakultas Teknologi Industri, Universitas Islam Indonesia dalam program Hibah Kolaborasi Dosen - Mahasiswa

CNF (*conjunctive normal form*) adalah salah satu bentuk dari formula proposisi yang memenuhi syarat sebagai formula dalam bentuk *normal form*. Sebuah formula disebut dalam bentuk *normal form* jika memenuhi syarat berikut[9]:

1. Hanya memuat operator alami, yaitu ∧, ∨, dan ¬.
2. Operator ¬ hanya boleh diterapkan terhadap proposisi (sub-formula yang berupa formula atomik)

CNF adalah salah satu bentuk formula yang termasuk formula dalam *normal form* karena memenuhi kedua syarat tersebut.

Secara formal, CNF didefinisikan seperti berikut ini. Sebuah formula *F* dikatakan dalam CNF jika formula *F* berbentuk:

$$F = C_1 \wedge C_2 \wedge C_3 \wedge \ldots \wedge C_n$$

Dengan

- $C_i = l_{i1} \vee l_{i2} \vee \ldots \vee l_{i,mi}$
- l_{ij} = p atau ¬p

C disebut sebagai klausa, *l* disebut sebagai literal dan p adalah proposisi. Dalam makalah ini, nilai benar dinyatakan dengan T dan nilai salah dinyatakan dengan F.

CONTOH 1
Formula-formula berikut merupakan formula dalam bentuk CNF. Untuk memudahkan pembacaan, setiap klausa yang terdiri dari lebih 1 literal ditulis dalam tanda kurung:

1. p ∧ q
2. p ∧ (q ∨ ¬r)
3. p ∧ q ∧ (¬p ∨ ¬q)
4. (p ∨ ¬q) ∧ q ∧ (¬p ∨ ¬r)
5. (p ∨ ¬q) ∧ (q ∨ ¬r) ∧ (r ∨ ¬p)
6. (p ∨ ¬q ∨ r ∨ ¬t)
7. (p ∨ ¬q ∨ r ∨ ¬t) ∧ (q ∨ t)

---------- ◊

B. SAT Problem dan SAT Solver

Salah satu tema penelitian yang paling banyak diminati di Logika Matematika adalah SAT Problem (*satisfiability problem*). Yang dimaksud dengan SAT Problem adalah menentukan apakah sebuah formula merupakan formula yang *satisfiabel* atau *unsatisfiabel*. Sebuah formula disebut *satisfiabel* jika ada kombinasi nilai kebenaran variabel pembentuknya yang membuat formula tersebut bernilai benar (T)[1].

CONTOH 2
Berikut ini adalah contoh penentuan sebuah formula sebagai formula yang *satisfiabel* atau tidak:

1. p ∧ q: *satisfiabel*, karena akan bernilai T jika p=T dan q=T.
2. p ∧ (q ∨ ¬r): *satisfiabel*, bernilai T jika p=T dan r=F.
3. p ∧ q ∧ (¬p ∨ ¬q) : tidak *satisfiabel*
4. (p ∨ ¬q) ∧ q ∧ (¬p ∨ ¬r) : *satisfiabel*, bernilai T jika salah satunya adalah q = T, p = F, dan r =F.

---------- ◊

Secara perangkat lunak, SAT Problem bisa diselesaikan dengan menggunakan SAT Solver. SAT Solver yang sudah dikembangkan dengan algoritma yang berbeda adalah sebagai berikut: WalkSat[10], Satz[11], GRASP[10], mChaff dan zChaff[12][13].

C. Algoritma DPLL

Algoritma DPLL adalah algoritma yang dikembangkan oleh Davis, Putnam, Logemann, dan Loveland untuk menyelesaikan SAT Problem dari formula dalam bentuk CNF[7]. Meskipun algoritma ini sudah sangat lama dikembangkan, namun sampai saat ini banyak SAT Solver yang dibuat berdasarkan algoritma ini. Beberapa SAT Solver yang disebutkan di subbab B adalah SAT Solver yang dibuat berdasarkan pengembangan dari Algoritma DPLL.

Algoritma DPLL dibuat berbasis aturan. Menurut [7] terdapat 5 aturan pada algoritma DPLL, yaitu *unit propagation*, *pure literal*, *decide*, *fail* (*unsatisfiable*), dan *backtrack*. Aturan-aturan ini akan diterapkan terhadap formula CNF sampai bisa diputuskan apakah formula tersebut *satisfiabel* atau *unsatisfiabel*. Setiap aturan bisa dijelaskan sebagai berikut:

1. *unit propagation*

   Aturan ini diterapkan jika terdapat klausa yang terdiri dari 1 literal. Proposisi pada literal itu ditugasi dengan nilai kebenaran sehingga bernilai T. Dengan penugasan ini semua klausa yang memuat literal yang sama akan bernilai T sehingga klausa-klausa tersebut bisa dihapus dari formula. Literal yang bernegasi dengan literal tersebut dihapus dari klausa yang memuatnya.

2. *pure literal*

   Aturan ini diterapkan jika terdapat sebuah literal yang hanya muncul dalam satu bentuk saja sementara negasi literal tersebut tidak muncul di klausa manapun. Literal tersebut ditugasi dengan nilai kebenaran sehingga bernilai T dan semua klausa yang memuat literal tersebut dihapus dari formula.

3. *decide*

   Aturan ini diterapkan jika semua literal muncul dalam 2 bentuk, misalnya p dan ¬p dan muncul dalam klausa yang berbeda. Yang dilakukan adalah menugasi salah satu literal dengan T, misal p, menghapus semua klausa yang memuat p, dan menghapus semua literal ¬p. Jika penugasan ini gagal (*fail*) maka diganti dengan penugasan negasinya (¬p).

4. *fail (unsatisfiable)*

Aturan ini diterapkan jika terdapat klausa yang kosong. Jika sebelumnya tidak ada aturan *decide* yang diberlakukan maka formula *unsatisfiable*.

5. *backtrack*

   Aturan ini diterapkan jika terjadi *fail* dan muncul di dalam *decide*. Yang dilakukan adalah kembali ke *decide* terdekat dan mengganti penugasan literal dengan negasinya. Misal, sebelumnya p yang ditugasi, maka langkah selanjutnya adalah menugasi ¬p.

Jika diperoleh formula yang kosong maka formula tersebut *satisfiable*.

CONTOH 3
Berikut ini adalah contoh penerapan algoritma DPLL untuk menyelesaikan formula (p ∨ ¬q) ∧ q ∧ (¬p ∨ ¬r) (Contoh 2 No. 4). Langkah-langkah penyelesaian adalah:

- *unit propagation* dengan q = T, diperoleh formula

  p ∧ (¬p ∨ ¬r)

- *unit propagation* dengan p = T, diperoleh ¬r
- *unit propagation* dengan r = F, diperoleh formula kosong sehingga bisa disimpulkan bahwa formula tersebut adalah *satisfiable*.

---------- ◊

III. HASIL DAN PEMBAHASAN

A. Desain

Sebuah formula CNF direpresentasikan dengan sebuah himpunan klausa. Sebuah klausa direpresentasikan dengan sebuah himpunan literal. Jadi sebuah formula akan direpresentasikan dengan himpunan dari himpunan literal.

Misalkan sebuah formula F adalah formula dalam bentuk CNF, yaitu

$$F = (l_{11} \vee l_{12} \vee \ldots \vee l_{1,m1}) \wedge (l_{21} \vee l_{22} \vee \ldots \vee l_{2,m2}) \wedge \ldots$$

$$\wedge (l_{n1} \vee l_{n2} \vee \ldots \vee l_{n,mn})$$

maka F dinyatakan dalam bentuk:

$$\{\{l_{11},l_{12},\ldots,l_{1,m1}\},\{l_{21},l_{22},\ldots,l_{2,m2}\}, \ldots ,\{l_{n1},l_{n2},\ldots,l_{n,mn}\}\}.$$

CONTOH 4
Representasi formula pada Contoh 1 adalah sebagai berikut:

1. {{p}, {q}}
2. {{p}, {q,¬r}}
3. {{p}, {q}, {¬p, ¬q}}
4. {{p, ¬q}, {q}, {¬p, ¬r}}
5. {{p,¬q}, {q, ¬r}, {r, ¬p}}
6. {{p,¬q, r, ¬t}}
7. {{p, ¬q, r, ¬t}, {q, t}}

---------- ◊

Dengan representasi formula CNF menggunakan himpunan dari himpunan literal, aturan-aturan dalam DPLL dinyatakan di bawah ini. Pada kasus ini, aturan *decide* dan *backtrack* digabung menjadi aturan *split* dan menambahkan aturan *sat*. Misalkan *F* adalah formula CNF dan *SAT(F)* adalah prosedur untuk menyelesaikan *F* maka aturan-aturan DPLL didefinisikan sebagai berikut:

1. *unit propagation*

$$\{l\} \in F \Rightarrow SAT(F') \text{ dengan}$$

$$F' = F \setminus \{C \mid l \in C \vee \neg l \in C\} \cup \{C \setminus \{\neg l\} \mid C \in F \wedge \neg l \in C\}$$

2. *pure literal*

$$C \in F, l \in C, \forall D \in F(\neg l \notin D)$$
$$\Rightarrow SAT(F \setminus \{E \mid E \in F, l \in E\})$$

3. *split*

$$C_1, C_2 \in F, l \in C_1, \neg l \in C_2$$
$$\Rightarrow SAT(F \cup \{\{l\}\}) \vee SAT(F \cup \{\{\neg l\}\})$$

4. *fail (unsatisfiable)*

$$\phi \in F \Rightarrow UNSAT$$

5. *sat*

$$F = \phi \Rightarrow SAT$$

Contoh 5 menunjukkan penerapan aturan-aturan tersebut untuk menyelesaikan formula pada Contoh 4.

CONTOH 5
Akan diselesaikan formula nomor 5 pada Contoh 4.

SAT({{p,¬q}, {q, ¬r}, {r, ¬p}})

⇒ SAT({{p,¬q}, {q, ¬r}, {r, ¬p},{p}}) ∨
  SAT({{p,¬q}, {q, ¬r}, {r, ¬p},{¬p}}) {split}

⇒ SAT({{q, ¬r}, {r}}) ∨
  SAT({{p,¬q}, {q, ¬r}, {r, ¬p},{¬p}}) {unit prop}

⇒ SAT({{q}}) ∨
  SAT({{p,¬q}, {q, ¬r}, {r, ¬p},{¬p}}) {unit prop}

⇒ SAT({}) ∨
  SAT({{p,¬q}, {q, ¬r}, {r, ¬p},{¬p}}) {unit prop}

⇒ SAT
  SAT({{p,¬q}, {q, ¬r}, {r, ¬p},{¬p}}) {sat}

⇒ SAT

---------- ◊

B. Implementasi

Implementasi SAT Solver dalam penelitian ini menggunakan bahasa pemrograman deklaratif Prolog. Aturan DPLL dinyatakan dalam sebuah predikat berikut ini:

```
sat(<formula>, <solusi>, <sat/unsat>)
```

Parameter `<formula>` adalah input, sedangkan parameter `<solusi>` dan `<sat/unsat>` adalah ouput.

Untuk setiap aturan kecuali split, definisi predikat adalah sebagai berikut:

```
sat(<formula>,<solusi>,<sat/unsat>) :-
   syarat(<aturan>,<formula>,<literal>),
   !,
   perbarui(<formula>,<literal>,
            <formula'>),
   sat(<formula'>,<solusi'>,<sat/unsat>.
```

Predikat syarat yang dinyatakan berikut

```
syarat(<aturan>,<formula>,<literal>)
```

adalah predikat yang menyatakan bahwa `<aturan>` dapat diterapkan terhadap `<formula>` untuk `<literal>`. Parameter `<solusi>` berbentuk himpunan (list) informasi solusi. Informasi solusi dinyatakan dalam bentuk `<aturan>(<literal>)`. Misalnya, informasi solusi untuk aturan unit propagation untuk literal p dinyatakan dengan `unit(p)`.

Sedangkan predikat perbarui yang dinyatakan berikut:

```
perbarui(<formula>,<literal>,<formula'>)
```

adalah predikat yang dengan parameter output `<formula'>` yang diperoleh dari `<formula>` dengan menghapus semua klausa yang memuat `<literal>` dan menghapus literal-literal yang merupakan negasi dari `<literal>`. Contoh deklarasi untuk aturan unit propagation dan pure literal ditunjukkan pada Gambar 1.

IV. Eksperimen

Dilakukan beberapa pengujian untuk menyelesaikan SAT Problem. Setiap SAT Problem disimpan dalam sebuah file teks. Format penulisan SAT Problem mengikuti aturan penulisan seperti di Kompetisi SAT (http://satcompetition.org). Pengujian dilakukan dengan menggunakan komputer Notebook dengan prosesor i5-6200U dan memori internal sebesar 4 GB.

Salah satu problem yang diujikan adalah permainan Sudoku. Permainan Sudoku dapat dinyatakan sebagai SAT Problem. Pada pengujian ini, penerjemahan Sudoku ke dalam SAT Problem menggunakan pengkodean seperti pada [14]. Dalam bentuk formula CNF, satu problem Sudoku diformulasikan dalam 729 variabel dan paling sedikit 8829 klausa. Problem Sudoku yang digunakan sebagai pengujian diambil dari Web Sudoku (https://www.websudoku.com). Pengujian hanya dilakukan terhadap problem Sudoku dalam 3 level, yaitu sedang (*medium*), sulit (*hard*), dan sangat sulit (*evil*).

SAT Solver dengan Prolog ini berhasil menyelesaikan semua Problem Sudoku yang diujikan. Semua problem dalam semua level diselesaikan dalam kisaran 30 detik. Jika langkah penyelesaian dibandingkan, problem Sudoku dengan level sangat sulit memerlukan langkah penyelesaian yang lebih banyak. Hal ini disebabkan oleh perlunya penerapan aturan *split* yang memerlukan *backtrack*. Penerapan aturan *split* dengan *backtrack* ini kadang-kadang juga diperlukan untuk level sulit. Sedangkan di level medium, aturan tersebut tidak pernah diterapkan.

SAT Solver ini juga diuji dengan problem-problem yang ada di Kompetisi SAT. Namun karena keterbatasan komputer yang digunakan, beberapa problem dalam kategori sedang harus diselesaikan dalam satuan menit. Spesifikasi komputer dalam pengujian ini tidak sebanding dengan komputer pada kompetisi SAT yang menggunakan prosesor Intel(R) Xeon(R) CPU E5-2609 dan memori internal sebesar 120GB.

Juga dilakukan pengujian dengan mengintegrasikan SAT Solver ini dengan sistem eksplorasi Formal Context secara kode program. Seperti disampaikan di bab Pendahuluan, SAT Solver digunakan oleh sistem eksplorasi Formal Context untuk menyelesaikan *entailment problem*. Dengan pengintegrasian ini, *entailment problem* yang sudah dikodekan ke dalam SAT Problem bisa dijadikan parameter input SAT Solver secara langsung tanpa melalui file teks.

Secara umum, SAT Solver ini meningkatkan unjuk kerja sistem tersebut. Namun peningkatan ini tidak terlalu besar karena kasus yang diselesaikan masih tergolong kecil. Sistem yang baru ini perlu diuji dengan problem yang lebih besar untuk melihat peningkatan unjuk kerjanya.

```
sat(F,[unit(Literal)|Solusi],Sat) :-
  syarat(unit,F,Literal),
  !,
  perbarui(F,Literal,FAksen),
  sat(FAksen,Solusi,Sat).

sat(F,[pure(Literal)|Solusi],Sat) :-
  syarat(pure,F,Literal),
  !,
  perbarui(F,Literal,FAksen),
  sat(FAksen,Solusi,Sat).
```

Gambar 1, Deklarasi aturan unit propagation dan pure literal

V. Kesimpulan

Telah berhasil dibangun sebuah SAT Solver menggunakan bahasa pemrograman deklaratif Prolog. Algoritma yang digunakan dalam SAT Solver ini adalah Algoritma DPLL. SAT Solver ini mampu menyelesaikan SAT Problem yang diujikan. Implementasi DPLL dengan pemrograman deklaratif pada dasarnya lebih cocok dibandingkan dengan konsep pemrograman yang lain karena algoritma DPLL dikembangkan dalam bentuk aturan (*rule*). Aturan adalah komponen dalam pemrograman deklaratif.

Unjuk kerja SAT Solver ini belum bisa dibandingkan dengan SAT Solver yang mengikuti kompetisi SAT. Perlu pengujian untuk menyelesaikan SAT Problem dengan kasus besar dan komputer dengan performa besar juga.

Untuk penelitian selanjutnya, perlu pengembangan lebih lanjut terhadap Algoritma DPLL dengan menyesuaikan konsep pemrograman deklaratif sehingga dapat meningkatkan unjuk kerja. Dalam penelitian ini, algoritma DPLL yang digunakan adalah algoritma dasar. Saat ini, semua SAT Solver yang mengikuti Kompetisi SAT dibangun dengan Algoritma DPLL yang sudah mengalami pengembangan namun dalam konsep pemrograman terstruktur atau pemrograman berarah obyek.

Referensi

[1] M. Huth dan M. Ryan, "Logic in Computer Science: Modelling and Reasoning about Systems", New York: Cambridge University Press, 2004.

[2] T. Hidayat, "Using Sat for Attribute Exploration of Formal Context with Constraint", Proceedings of International Conference on Information Technology, 2013.

[3] T. Hidayat,, "Attribute exploration of many-valued context with SAT", Proceedings of 10th WSEAS international conference on Computational Intelligence, Man-Machine Systems and Cybernetics, pp. 33-37, 2011.

[4] T. Hidayat, "Association Rules of Web Mining with Background Knowledge of Web Structure", Proceedings of The 14th International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases, pp. 215, 2015.

[5] J.M. Howe dan A. King, "A Pearl on SAT Solving in Prolog", dalam: M. Blume, N. Kobayashi, G. Vidal. (eds) *Functional and Logic Programming. FLOPS 2010*, *Lecture Notes in Computer Science*, vol 6009, Heidelberg: Springer.

[6] M. Codish, V. Lagoon, dan P. Stuckey, "Logic programming with satisfiability", Theory Pract. Log. Program. Vol 8, pp. 121-128, 2008.

[7] R. Nieuwenhuis, dkk, "Solving SAT and SAT Modulo Theories: From an Abstract Davis–Putnam–Logemann–Loveland Procedure to DPLL(T)", Journal of the ACM, Vol. 53, No. 6, pp. 937–977, November 2006.

[8] Y. Hamadi, S. Jabbour, dan L. Saïs, "What we can learn from conflicts in propositional satisfiability". Annals of Operations Research, 240(1), pp. 13-37, 2016.

[9] T. Hidayat, "Logika Proposisi", Yogyakarta: Dar Fiqrin, 2014.

[10] J.P. Marques-Silva, dan K.A. Sakallah, "GRASP: A New Search Algorithm for Satisfiability", Proceedings of International Conference on Computer-Aided Design, pp. 220-227, 1996.

[11] C.M. Li, dan Anbulagan, "Heuristics Based on Unit Propagation for Satisfiability Problems", Proceedings of IJCAI, 366-371, 1997

[12] M. Moskewicz, dkk. "Chaff: Engineering an Efficient SAT Solver", 39th Design Automation Conference (DAC 2001), ACM 2001.

[13] Y. Vizel, G. Weissenbacher, dan S. Malik, "Boolean Satisfiability Solvers and Their Applications in Model Checking", Proceedings of the IEEE., 103, 2015.

[14] I. Lynce dan J. Ouaknine, "Sudoku as a SAT Problem", Proc. of the 9th International Symposium on Artificial Intelligence and Mathematics, AIMATH 2006, Heidelberg: Springer, 2006.