

SAT and MAX-SAT Encodings for Triangle Packing

Tom Lotgering, 1183052

1. Introduction

The vertex-disjoint triangle packing problem is an NP-hard problem with applications in, among others, the matching of patient-donor pairs for kidney transplantation. This paper shows that this problem can in many cases be solved efficiently by translating the problem to SAT, and solving the encoded problem with a SAT solver.

Section 1 introduces the Triangle Packing and SAT problems. In section 2 we discuss various SAT and Max-SAT encodings for Triangle Packing. Section 3 describes the experiments performed on these encodings and the solvers used for those experiments. Section 4 shows the results of these experiments, and section 5 contains the conclusion.

Triangle Packing

Given a graph $G = (V, E)$. Let a triangle T_i be a triplet (u, v, w) such that $u, v, w \in V$ and $(u, v), (v, w), (w, u) \in E$. A selection of triangles is vertex-disjoint if for any two triangles T_i, T_j in the selection (where $i \neq j$) there exists no $v \in V$ such that $v \in T_i \wedge v \in T_j$. The Triangle Packing decision problem is then to select $\lfloor |V|/3 \rfloor$ vertex-disjoint triangles, while the optimization problem is to select the maximum number of vertex-disjoint triangles [4].

Satisfiability

A satisfiability (SAT) problem is a decision problem, whose instance is a Boolean expression consisting of a set X of variables x , and a set C of clauses (constraints) consisting of literals, where a literal is a variable or the negation of a variable in X . The problem is typically in conjunctive normal form (CNF) where every clause is a disjunction of literals, and the problem is a conjunction of clauses.

2. Encodings

We've implemented two SAT encodings for Triangle Packing, as well as Max-SAT variations on these encodings.

SAT Encodings

For the normal satisfiability variant, the problem instance is solved iff all clauses are satisfied.

We've implemented two satisfiability encodings for Triangle Packing. They both consist of a set of clauses to force the selection of triangles (the "positive" clauses) and a set of clauses to prevent the selection of overlapping triangles (the "negative" clauses). Encoding 1 and 2 differ only in their implementation of the negative clauses.

Encoding 1

Let $T = \{T_1 \dots T_m\}$ be the set of triangles in G . Let the triangle span $TS(v)$ be the set of triangles that contain vertex v .

Variables:

$$\{x_i | T_i \in T\}$$

Positive Clauses:

$$\bigwedge_{v \in V} \left(\bigvee_{T_i \in TS(v)} x_i \right)$$

Negative Clauses:

$$\bigwedge_{T_i, T_j \in TS(v) | i < j, v \in V} (\neg x_i \vee \neg x_j)$$

Encoding 2

Since encoding 1 uses a number of negative clauses that is quadratic in the size of $TS(v)$ to ensure that at most one triangle of each overlapping set of triangles is selected, a more compact encoding of this at most one constraint is desirable.

One method to reduce the number of clauses required for the at-most-one constraint is the “ladder encoding” described in [5], which defines new variables $y_1 \dots y_{n-1}$ and adds “ladder validity clauses” $\neg y_{i+1} \vee y_i$ and “channeling clauses” expanded from $x_i \leftrightarrow (y_{i-1} \wedge \neg y_i)$, thereby adding $O(n)$ new variables but using only $O(n)$ clauses (when encoding at-most-one over a set of n objects). A more efficient encoding based on the ladder encoding is the AMO encoding presented below, whose author is regrettably unpublished.

$$AMO(x_1, \dots, x_k) = \begin{cases} \text{if } k \leq 4: & \bigwedge_{i < j} (\neg x_i \vee \neg x_j) \text{ if } k \leq 4 \\ \text{otherwise: } & AMO(x_1, x_2, x_3, z) \wedge AMO(\neg z, \dots, x_k) \end{cases}$$

This encoding will use approximately $\frac{n}{2} * \binom{4}{2} = 3n$ clauses, because (except for the first and last AMO tuple) a tuple will contain a “payload” of 2 variables, hence $\frac{n}{2}$ tuples for large n , and a tuple (except for the last) consists of 4 literals, and every combination within a tuple is excluded, hence $\binom{4}{2}$. The number of variables introduced is $n/2$. Tuple sizes other than 4 are possible, but introduce either more clauses or more variables. For example, for a tuple size of 5, the expected number of clauses is $\frac{n}{3} * \binom{5}{2} = 3\frac{1}{3}n$ and the number of variables is $n/3$. For a tuple size of 3, the expected number of clauses is $\frac{n}{1} * \binom{3}{2} = 3n$ but the number of variables introduced is n , as this is effectively a ladder encoding.

Using the given definition of AMO encoding 2 is defined by:

Variables:

$$\{x_i | T_i \in T\} \cup \{x_q | z_q \in AMO(v), v \in V\}$$

Positive Clauses: see encoding 1.

Negative Clauses:

$$\bigwedge_{v \in V} AMO(TS(v))$$

Partial MAX-SAT Encodings:

Both encodings described for SAT have a Partial Max-SAT equivalent. There are two variations to these two encodings; one a straight-forward translation, the other using the unique capabilities offered by Max-SAT. The variations differ only in their implementation of the “positive clauses” as defined above.

With both encodings and both variations, the positive clauses are soft constraints, and the negative clauses are hard constraints. The positive clauses therefore are assigned weight 1, and the negative clauses are assigned a weight B greater than the sum of all positive clauses, in this case $B = \text{number of variables} + 1$.

Vertex-based Positive Clauses

This Max-SAT encoding is a straight-forward translation from the SAT encoding (which can be either encoding 1 or 2). The benefit provided by using the Max-SAT version is that for problems without perfectly optimal solution imperfect solutions may be found that assign an optimal number of triangles, while still remaining valid solutions.

Both positive and negative clauses are equivalent to those in the SAT encoding, except for the addition of weights. As mentioned, the positive clauses are assigned weight 1 and the negative constraints get weight $(\text{number of variables} + 1)$. A problem encoded as such is satisfied when all clauses are satisfied.

The positive clauses are therefore:

$$\bigwedge_{v \in V} \left(\bigvee_{T_i \in TS(v)} x_i \right) \{ \text{weight} = 1 \}$$

Triangle-based Positive Clauses

The second variation recognizes that a much simpler alternative to the normal SAT encoding’s positive clauses may be used for partial Max-SAT. Since the goal is only to minimize the cost rather than to satisfy each clause, an encoding that uses mutually exclusive clauses may be valid.

As with the previous variation, the negative constraints get weight $(\text{number of variables} + 1)$. This variation however has simpler positive clauses. Its positive clauses are simply a list of triangles, each with weight 1. Since many of these triangles overlap and the choice of overlapping triangles is prevented by the negative clauses, obviously these cannot all be satisfied, but if an optimal number of triangles is selected (and therefore clauses satisfied), it is still an optimal solution. The positive clauses are therefore:

$$\bigwedge_{T_i \in T} (x_i) \{ \text{weight} = 1 \}$$

Since the positive clauses can no longer each be satisfied under this encoding, the problem instance is solved if the cost (the sum of the weight of the unsatisfied clauses) satisfies the equation $|T| - cost = \lfloor |N|/3 \rfloor$, where $|T| - cost$ is the number of assigned triangles when none of the hard constraints are violated.

(Un)Satisfiability VS Triangle Packing

With these encodings, there is no direct relationship between triangle packing no-instances and satisfiability no-instances. Imagine a graph with only one possible triangle, and 6 vertices. See Figure 1. Using any of the mentioned unweighted encodings, this will translate (when trimmed) to the degenerate case with only one variable, and only the clause (1). Obviously, this is satisfiable. However, since the number of triangles assigned is less than $\lfloor |V|/3 \rfloor = 2$, this is a TrianglePacking no-instance. When using the partial max-sat encoding with Triangle-based positive clauses and a max-sat solver, this case is detected by verifying that $|T| - cost = \lfloor \frac{|V|}{3} \rfloor$. The normal SAT solvers operating on an unweighted encoding, or a max-sat solver operating on an encoding with the Vertex-based positive clauses, won't detect this case.

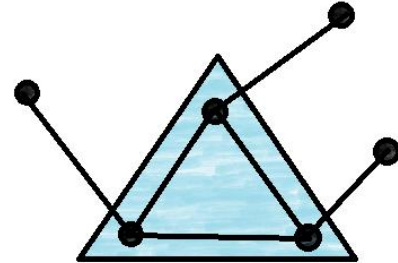


Figure 1: TrianglePacking no-instance

This degenerate case occurs because the graph contains $3 - (|V| \bmod 3)$ or more vertices which are not part of any triangle. $|V| \bmod 3$ is the number of vertices that need not be assigned in Triangle Packing problems, since those take require only $\lfloor \frac{|V|}{3} \rfloor$ triangles to be packed. Vertices which are not part of any triangle are not represented by normal encoding 1 & 2, because for these vertices the positive clause $(\bigvee_{T_i \in TS(v)} x_i)$ is empty. This causes the SAT encoding to ignore them.

Any graph which satisfies $|\{v | TS(v) = \emptyset, v \in V\}| \geq 3 - (|V| \bmod 3)$ is a Triangle Packing no-instance. During encoding it can be checked if G satisfies this property, in which case the encoder could return a trivial unsatisfiable instance, to ensure that $yes_{SAT} \leftrightarrow yes_{TrianglePacking}$ and $no_{SAT} \leftrightarrow no_{TrianglePacking}$ when using non max-sat solvers or the Vertex-based positive clauses.

3. Experiments

The performance of the presented encodings was tested on random graphs using a variety of solvers.

Solvers

Several solvers were used to investigate the performance of the various encodings.

Plingeling

Plingeling is the result of several generations of improvement of Conflict Driven Clause Learning (CDCL) solvers. Many successful complete SAT solvers are based on the branch and backtracking algorithm called the Davis Putnam Logemann Loveland (DPLL) algorithm. CDCL algorithms expand on DPLL by adding a pruning technique called learning.

DPLL solvers construct a solution by incrementally assigning (branching) variables, and backtracking on these decisions when the problem becomes unsatisfiable (when a clause is falsified by the decision) for this partial assignment. When a decision is made on a branching variable, this creates a

new “decision level”, and any variable assignments resulting from the decision (due to *unit propagation*) share that decision level. While a basic DPLL solver will backtrack one decision[2] level as a result of a conflict, a CDCL solver uses more advanced analysis of the *implication graph* to determine the actual reasons for the conflict and backtrack more than one decision level [3].

Plingeling improves on previous CDCL solvers by interleaving preprocessing algorithms with the CDCL algorithm and using efficient data structures, as well as making use of multiple processing cores [6].

Experiments with Plingeling used only the (unweighted) SAT encoding, since it is not a Max-SAT solver.

MSUnCore

MSUnCore (Maximum Satisfiability with UNSatisfiable COREs) is an unsatisfiability-based Max-SAT solver. The unsatisfiability-based solver approach consists of identifying unsatisfiable sub-formulas, relaxing each clause in each unsatisfiable sub-formula, and adding a new constraint requiring exactly one relaxation variable to be relaxed in each unsatisfiable sub-formula. MSUnCore uses PicoSAT, a CDCL SAT solver like Plingeling, for the iterative identification of unsatisfiable sub-formulas [1].

Unsatisfiability based solvers use the ability of (certain) SAT solvers to provide a certificate of unsatisfiability in the form of an unsatisfiable core (sub-formula) to solve the Max-SAT problem optimally. Unsatisfiability-based algorithms start by using an underlying SAT solver to solve the problem. If the solver returns SATISFIABLE, the SAT solution is returned as the Max-SAT solution. If the solver returns UNSATISFIABLE, the algorithm will relax the clauses in the unsatisfiable core returned by the SAT solver by adding relaxation variables which, when set to true, satisfy the clause. This process is repeated until the problem is satisfied. The resulting solution can be shown to be optimal, since each relaxed clause eliminates at most one UNSAT core [7].

Experiments were ran using MSUnCore with both weighted and unweighted encodings.

SAPS (UBCSAT)

Scaling and Probabilistic Smoothing (SAPS) is a Stochastic Local Search (SLS) algorithm. An SLS solver is incomplete: it does not guarantee that its solution is optimal; a problem may be satisfiable if the algorithm is unable to find a solution that satisfies all clauses. When the problem is satisfiable, an SLS solver may often find a solution effectively.

The general strategy of an SLS solver is to start with an initial complete assignment of truth values to all variables in the given formula and in each step flip the truth assignment of one variable, with the purpose of minimizing an objective function: typically the number of clauses unsatisfied by the variable assignment.

SAPS uses an SLS method called Dynamic Local Search (DLS). DLS strategies are based on the idea of modifying the evaluation function in order to prevent the search from getting stuck in local minima. They typically associate weights with the clauses of the given formula, which are modified during the search process, and the objective becomes to minimize the total weight, rather than simply the number of falsified clauses. There are several variants of this scheme, including the Exponentiated Sub-Gradient method (ESG), which SAPS is closely related to.

ESG for SAT works as follows: The search is initialized by randomly chosen truth values for all propositional variables in the input formula, F , and by setting the weight associated with each clause in F to one. Then, in each iteration, a weighted search phase followed by a weight update is performed. During the weighted search phase a series of greedy variable flips are performed on variables selected at random from the set of all variables that appear in currently unsatisfied clauses. The clause weights are updated after the search phase based on their satisfaction status. The algorithm terminates when a solution is found or a maximal number of iterations (determined by the cutoff) have been performed.

SAPS differs from ESG in the way it implements weight updates: SAPS performs the computationally expensive weight smoothing probabilistically and less frequently than ESG, leading to a reduction in the time complexity of the weight update procedure without increasing the number of variable flips required for solving a given SAT instance [8].

Settings used:

- Cutoff: 10.000.000 (default = 100.000)

Experiments were ran using UBCSAT’s implementation of SAPS on weighted encodings.

Graphs

For these experiments I used Erdos-Renyi random graphs of size N , which are constructed by allowing an edge between any pair of nodes to occur with probability p .

4. Results

Triangle Packing Difficulty

The following results display the properties of the triangle packing problem, when translated to SAT.

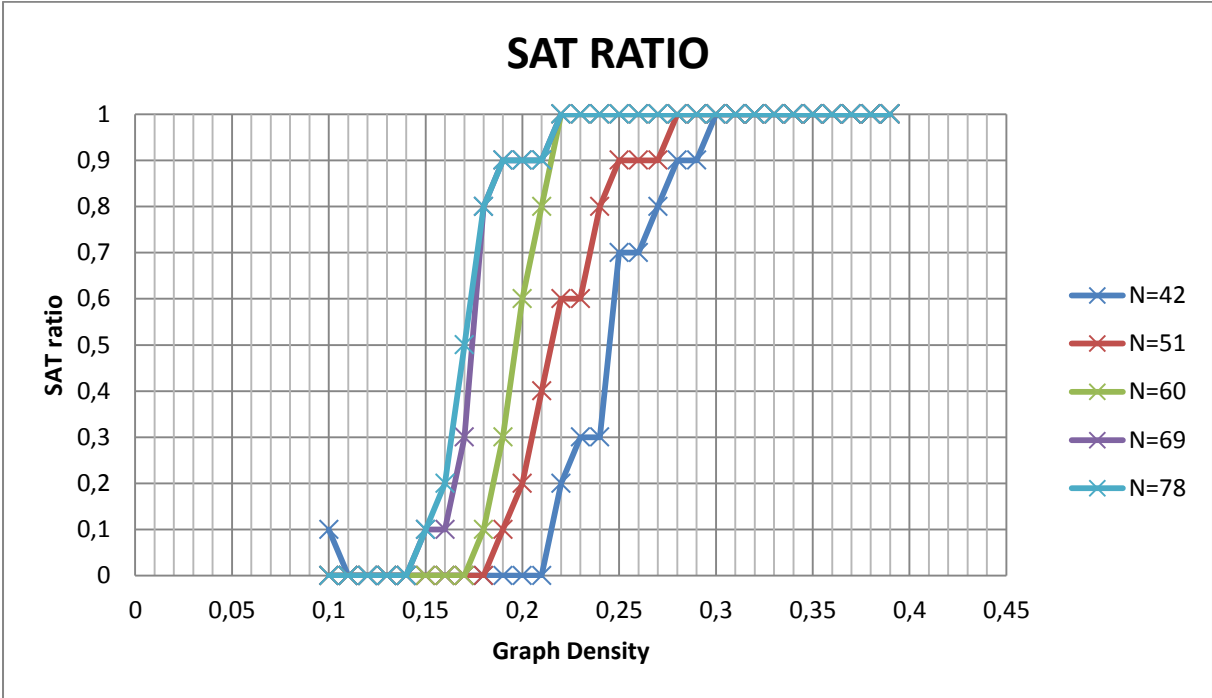


Figure 2: SAT Ratio. N=number of vertices

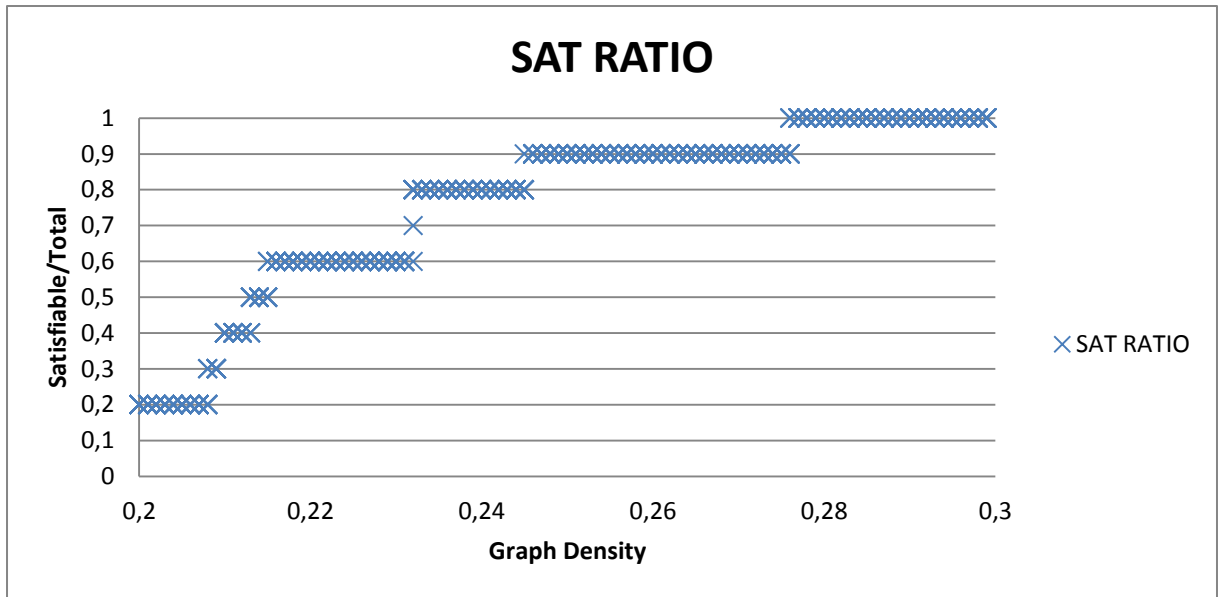


Figure 3: SAT Ratio N=51

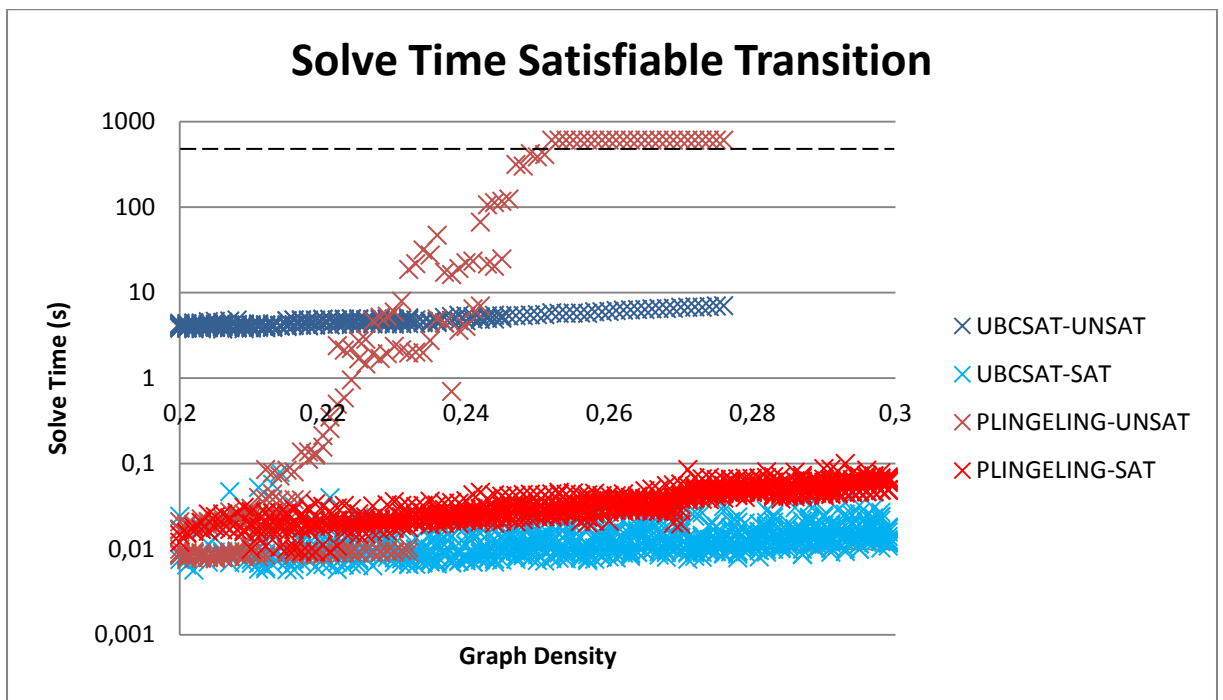


Figure 4: Solve Time SAT Transition. N=51. Enc 2 & C.Pos (UBCSAT). Dashed line represents 600 sec timeout.

MSUnCore

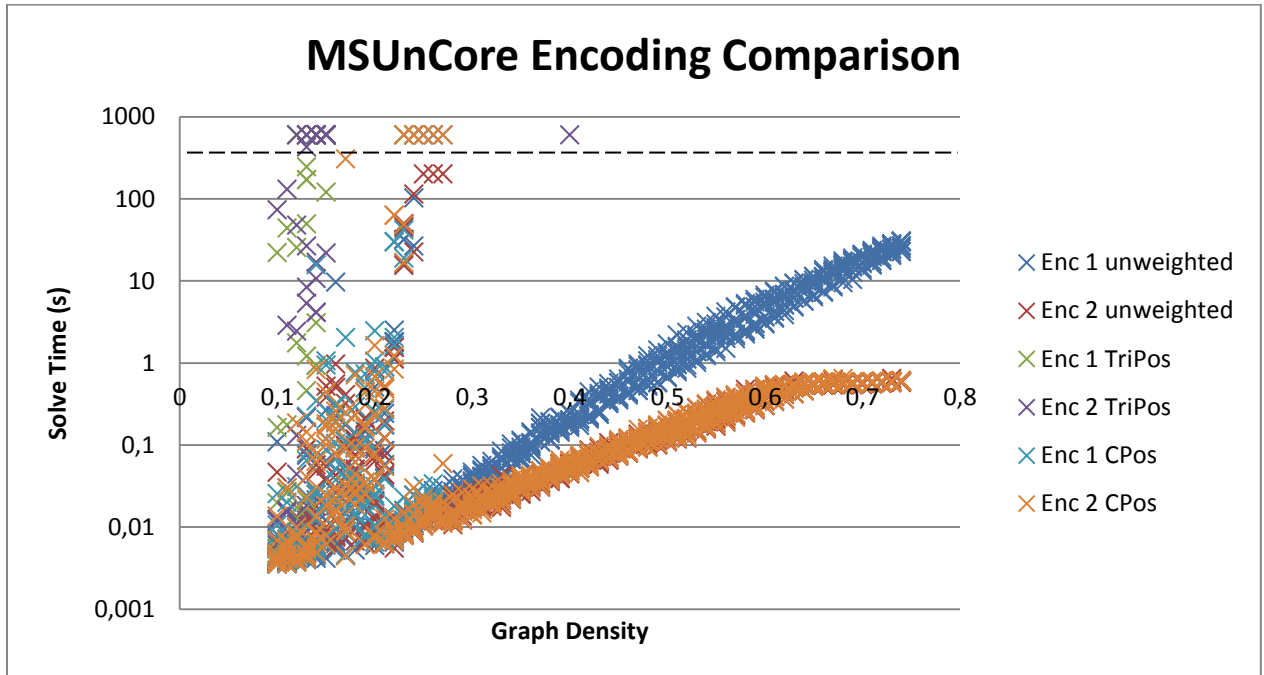


Figure 5: MSUnCore Encoding Comparison. N=51. Dashed line represents 600 sec timeout.

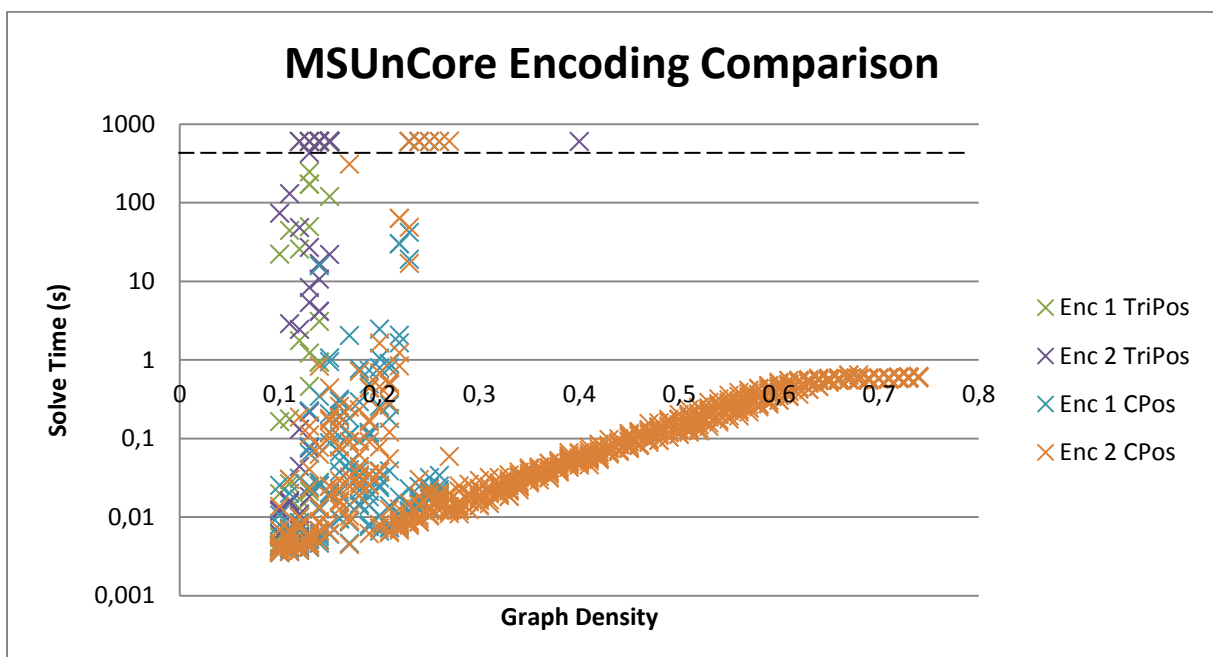


Figure 6: MSUnCore Weighted Encodings. N=51. Dashed line represents 600 sec timeout.

PLINGELING

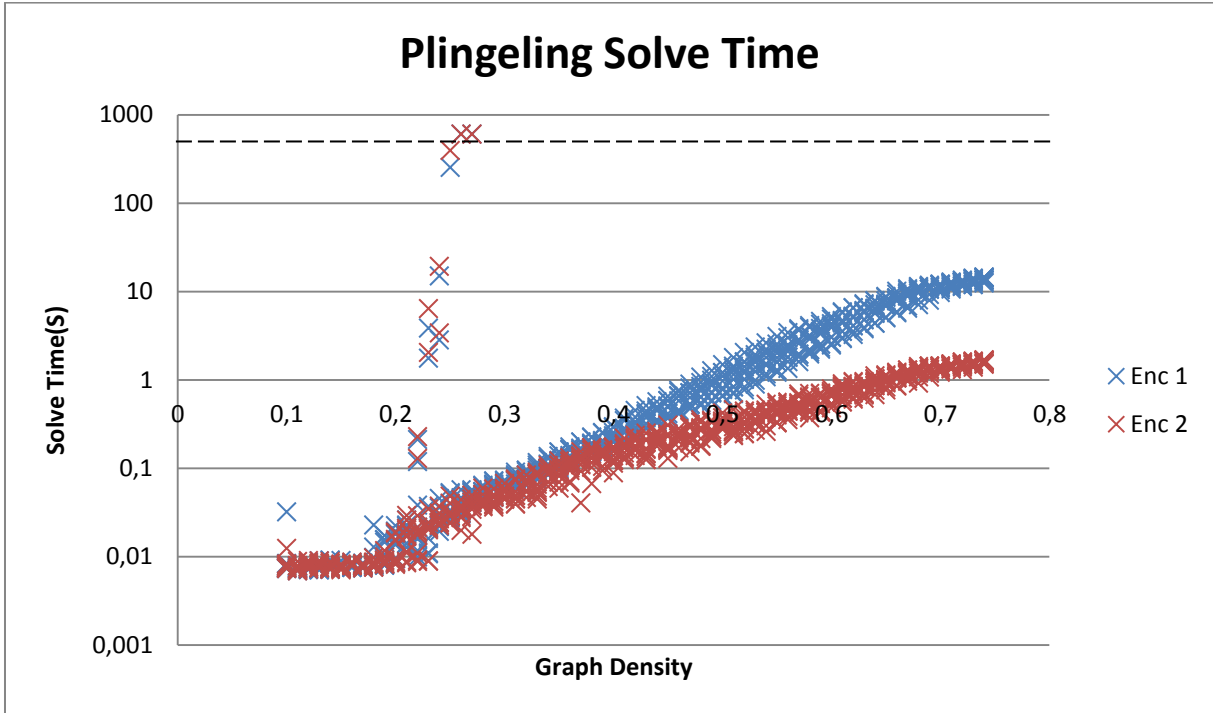


Figure 7: Plingeling Encoding Comparison. N=51. Dashed line represents 600 sec timeout.

UBCSAT

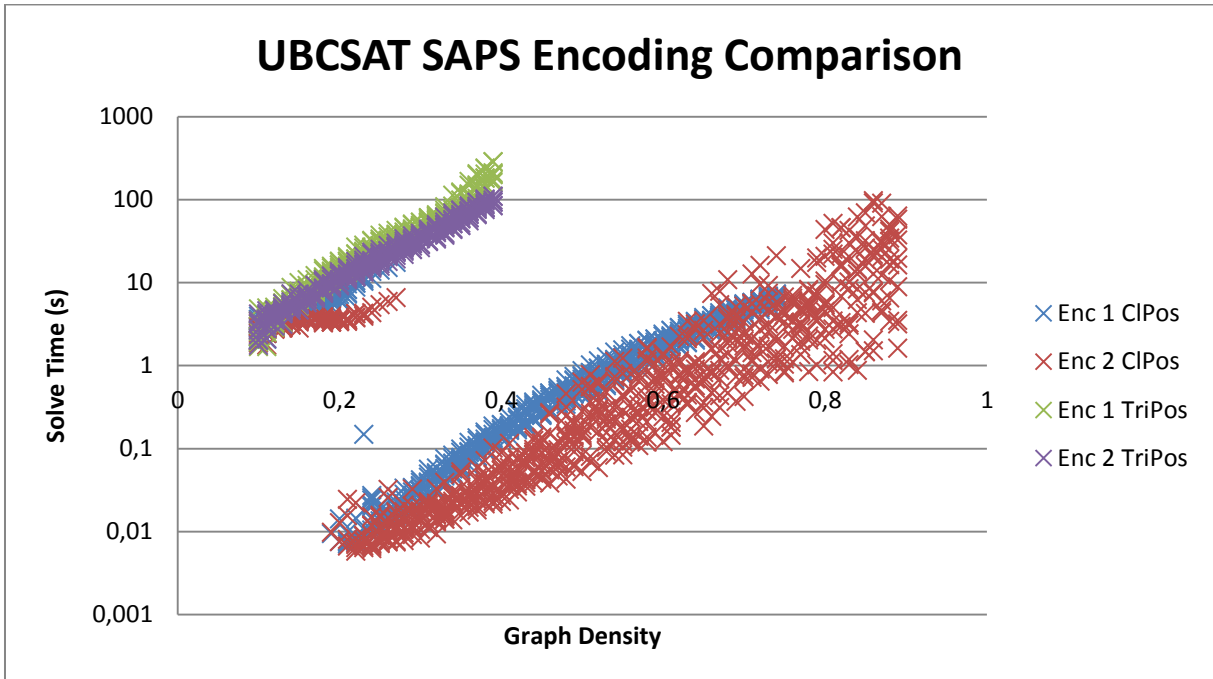


Figure 8: UBCSAT Encoding Comparison. N=51

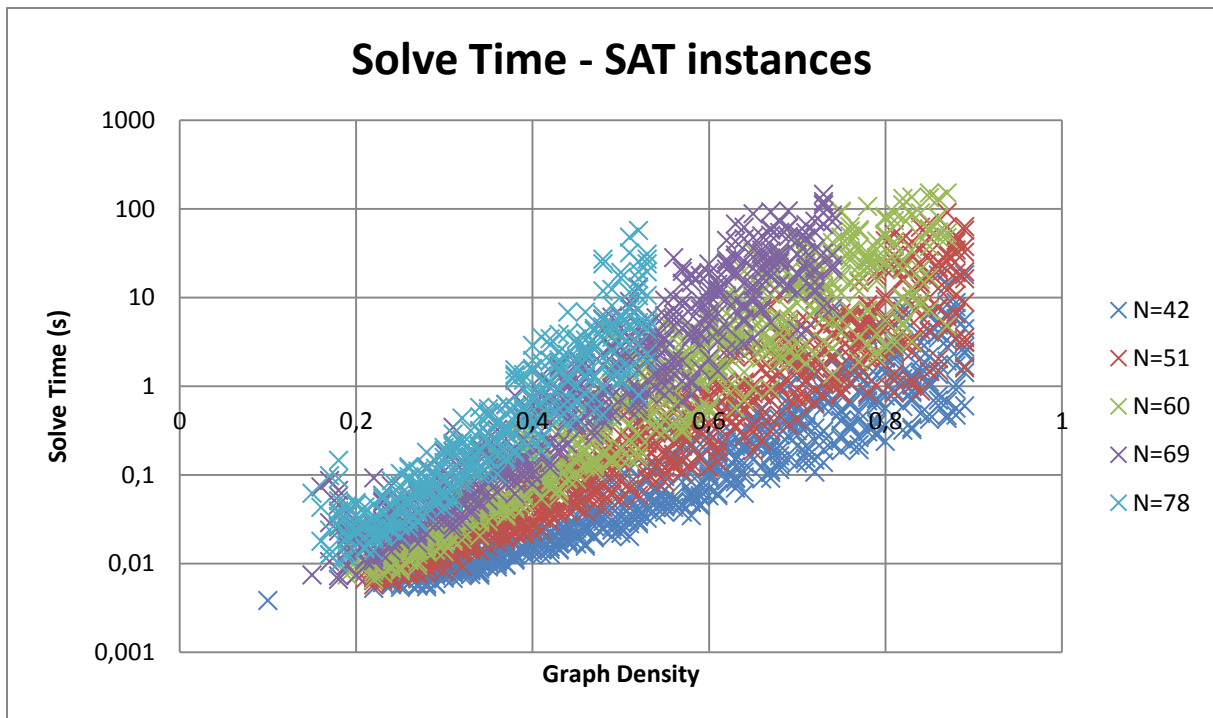


Figure 9: UBCSAT solve time. Enc. 2 & C.Pos

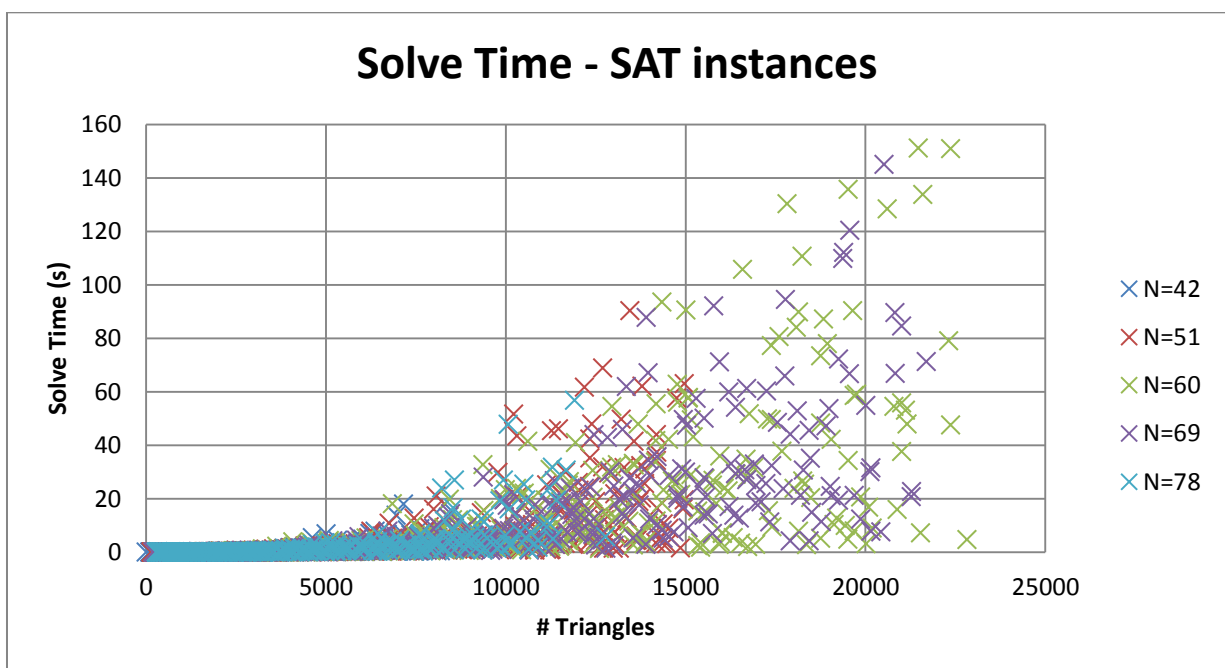


Figure 10: UBCSAT - SAT solve time. Enc. 2 & C.Pos

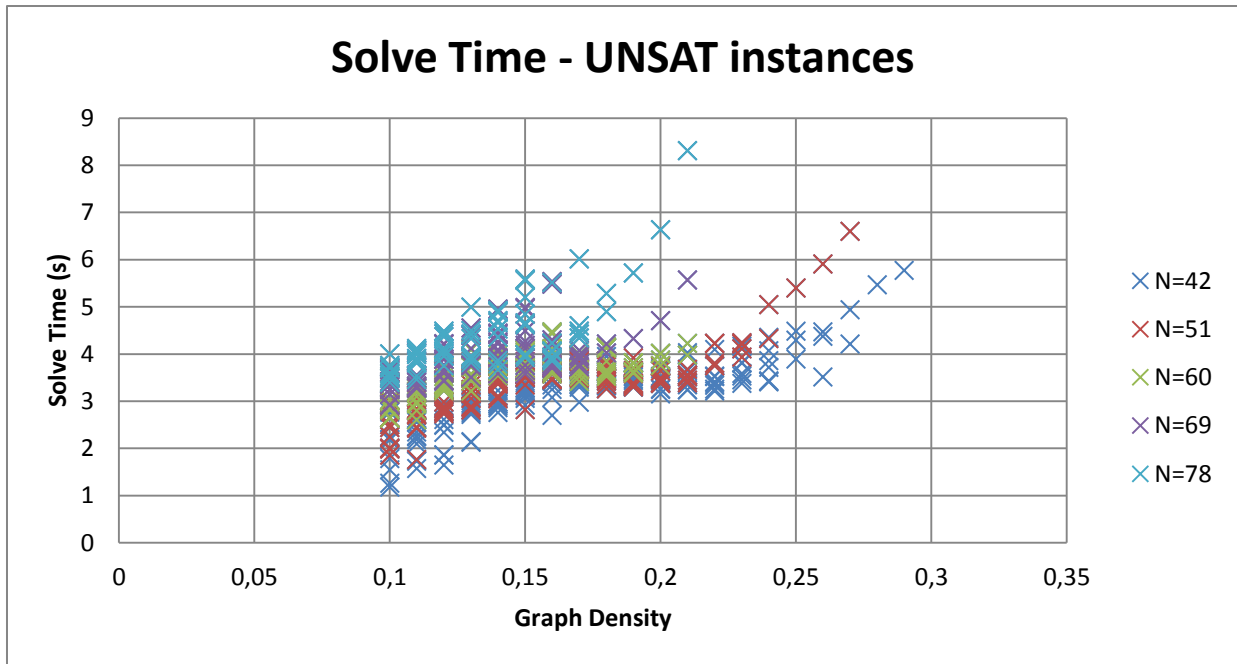


Figure 11: UBCSAT - UNSAT solve time. Enc. 2 & C.Pos

Solver Comparison

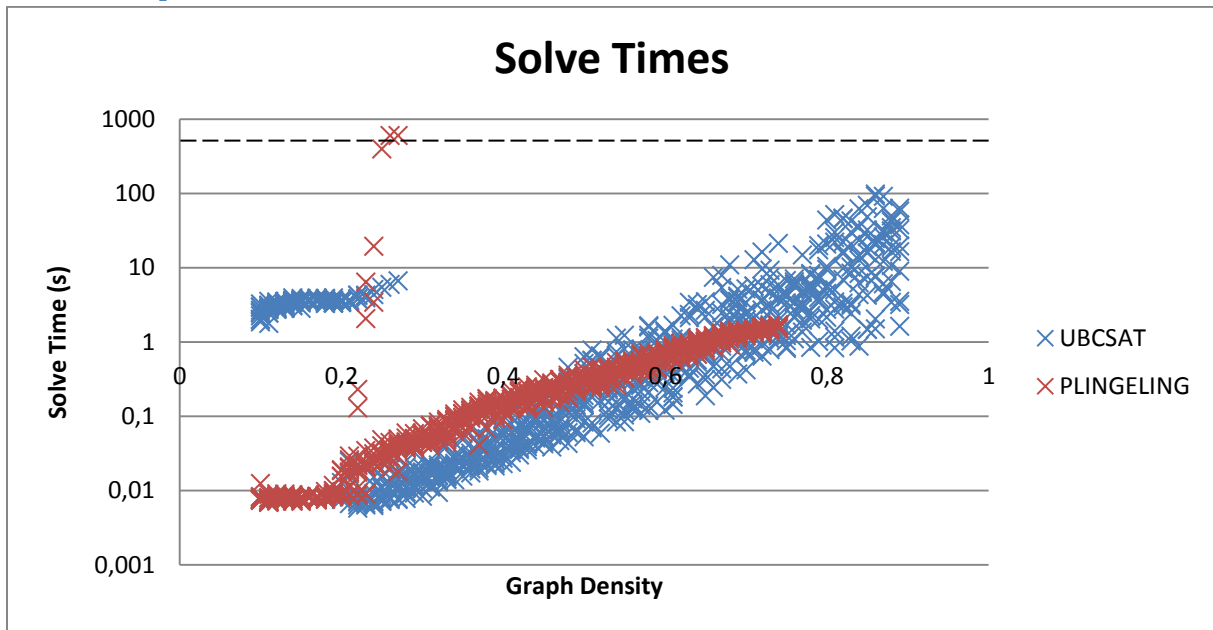


Figure 12: Solver Comparison. N=51. Enc 2 & C.Pos (UBCSAT) . Dashed line represents 600 sec timeout.

5. Conclusion

At the lowest graph densities, few triangles can be found and therefore they are unsatisfiable. At high graph densities, there are so many potential triangles that finding a non-overlapping selection that covers all vertices is trivial, and therefore these instances are trivially satisfiable. There is a region where the problem changes from mostly unsatisfiable, to mostly satisfiable. Figure 2 shows the position of the transition between unsatisfiable and satisfiable instances, as a function of graph density, and figure 3 shows this transition in detail when the graph has size 51. Figure 4 shows that some solvers, especially complete solvers such as Plingeling, require significantly more time (it

surpasses the timeout of, in this case, 10 minutes and therefore does not yield any result) for some problems that are on this transition. Solvers using local search algorithms, such as UBCSAT using SAPS, spend more time on unsatisfiable instances in general, but do not struggle specifically around the transition.

Figure 7 shows the overall performance of Plingeling. It performs well outside the satisfiability transition, even though the solve time grows rapidly as a function of graph density (it appears exponential, although it is polynomial in the number of potential triangles). Encoding 2 clearly performs better than encoding 1 outside this transition, but is no better or even worse on the harder instances. As figure 4 showed, it does not perform well within the transition region; even though the time-limit for the graph is set to only 10 minutes, it fails to find a solution when given several hours, with either encoding.

MSUnCore is capable of solving Max-SAT problem with or without clause weights. When using MSUnCore with unweighted encodings, it performs similarly to Plingeling. It does well outside the satisfiability transition, but will time-out on some of the hardest instances. For those instances it does solve efficiently, encoding 2 outperforms encoding 1, especially at the highest graph densities (see figure 5). MSUnCore performs worse on weighted problems than on unweighted problems, even if these weights are uniform and used only to distinguish between hard and soft constraints. For the weighted encoding with the same Vertex-based positive constraints as the unweighted encoding (Enc 1 / 2 CPos in the graph), it performs slightly worse (compared to its unweighted version) on the unsatisfiable instances, and similarly on the satisfiable ones. When using the weighted encoding with Triangle-based positive constraints (Enc 1 / 2 TriPos in the graph) however, it fails to solve any non-trivial problem in reasonable time (see figure 6).

SAPS-based UBCSAT performs better than any of the other solvers on the problems in the satisfiability transition, and does well on the other problems, when using the Vertex-based positive constraints: it is able to solve all problems in reasonable time, and achieves the same result as Plingeling. Encoding 2 usually performs better than encoding 1, although it has a higher variance. Like MSUnCore SAPS performs much worse on the Triangle-based positive clauses. When using Triangle-based positive clauses, and when using the same cutoff of 10.000.000, it does not spend more time on the unsolvable instances than when using the Vertex-based positive clauses (since the algorithm only terminates once this cutoff is reached or an optimal solution is found), but it fails to solve the satisfiable instances which were solved using the other encoding (see figure 8). With an increased cutoff (100.000.000) and the same time limit (10 minutes) used for testing the other solvers, it still fails to solve any satisfiable instances in reasonable time.

UBCSAT performs no worse on most problems than the other solvers (although Plingeling scales slightly better with graph density), but significantly better on the hardest instances, which the other solvers sometimes even fail to decide at all. The reason for this is, of course, that UBCSAT does not attempt to decide the problem but simply concedes “unknown” if it fails to find a solution quickly, whereas the other algorithms will continue to run in an attempt to find definitive proof of unsatisfiability.

With all solvers, encoding 2 performs better than encoding 1 on satisfiable instances, with benefit increasing with graph density and size. For such graphs, the quadratic growth of encoding 1's negative clauses is prohibitive, and encoding 2 replaces this with only linear growth.

For weighted encodings, those using Triangle-based positive clauses perform much worse than those using the same Vertex-based positive clauses as the unweighted encoding. Neither UBCSAT nor MSUnCore manages to solve any of the satisfiable instances when using these encodings. The reasons for this are twofold. First of all, with these Triangle-based positive clauses a valid solution must necessarily violate a large portion of the soft constraints, while with the Vertex-based positive clauses a solution that covers all vertices will also satisfy all constraints. Therefore using the Vertex-based positive clauses, the solver itself is able to decide when it has found an optimal solution, while with the other positive constraints this is decided afterwards by comparing the number of falsified clauses to the number of triangles. Secondly, the vertex-based positive clauses explicitly describe the structure of the problem by giving for each vertex the list of triangles it participates in as a single clause, which benefits all solvers in some way, although indirectly.

6. References

- [1] J. Marques-Silva. "The MSUnCore Max-SAT Solver." *SAT 2009 competitive events booklet*, page 151, 2009.
- [2] J.P. Marques-Silva, K.A. Sakallah, "GRASP: A Search Algorithm for Propositional Satisfiability", *IEEE Transactions on Computers*, pages 506-521, 1999.
- [3] L. Zhang, C.F. Madigan, M.H. Moskewicz, S. Malik, "Efficient Conflict Driven Learning in a Boolean Satisfiability Solver", *ACM Press*, 2001.
- [4] P. Prosser, "Triangle Packing with Constraint Programming", *9th International Workshop on Constraint Modelling and Reformulation*, pages 1-15, 2010.
- [5] A. Biere, M. Heule, H. Van Maaren, T. Walsh, "Handbook of Satisfiability", *ch. 2*, *IOS Press*, 2009.
- [6] A. Biere, "Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT race 2010".
- [7] Z. Fu, S. Malik, "On Solving The Partial MAX-SAT Problem", *Theory and Applications of Satisfiability Testing*, pages 252-265, *Springer*, 2006.
- [8] F. Hutter, D. Tompkins, H. Hoos. "Scaling and Probabilistic Smoothing: Efficient Dynamic Local Search for SAT", *Principles and Practice of Constraint Programming*, pages 241-249, *Springer*, 2006.