09491 Abstracts Collection
Graph Search Engineering
— Dagstuhl Seminar —

Lubos Brim[1], Stefan Edelkamp[2], Eric Hansen[3] and Peter Sanders[4]

[1] Masaryk University, CZ
brim@fi.muni.cz
[2] Universität Bremen, DE
edelkamp@tzi.de
[3] Mississippi State Univ., US
hansen@cse.msstate.edu
[4] KIT Karlsruhe, DE
sanders@ira.uka.de

Abstract. From the 29th November to the 4th December 2009, the Dagstuhl Seminar 09491 "Graph Search Engineering " was held in Schloss Dagstuhl – Leibniz Center for Informatics. During the seminar, several participants presented their current research, and ongoing work and open problems were discussed. Abstracts of the presentations given during the seminar as well as abstracts of seminar results and ideas are put together in this paper. The first section describes the seminar topics and goals in general. Links to extended abstracts or full papers are provided, if available.

Keywords. Model Checking, Artificial Intelligence, AI Planning, State Explosion Problem, Error Detection, Protocol Analysis, Software Verification and Validation, Heuristics, Pattern/Abstraction Databases, I/O Efficient Search, Solid State Disks, GPU

1 Introduction

Graph Search algorithms and their variants play an important role in many branches of computer science. All use duplicate detection in order to recognize when the same node is reached via alternative paths in a graph. This traditionally involves storing already-explored nodes in random-access memory (RAM) and checking newly-generated nodes against the stored nodes. However, the limited size of RAM creates a memory bottleneck that severely limits the range of problems that can be solved with this approach. Although many clever techniques have been developed for searching with limited RAM, all eventually are limited in terms of scalability, and many practical graph-search problems are too large to be solved using any of these techniques.

Over the past few years, several researchers have shown that the scalability of graph-search algorithms can be dramatically improved by using external memory, such as disk, to store generated nodes for use in duplicate detection. However, this requires very different search strategies to overcome the six orders-of-magnitude difference in random-access speed between RAM and disk. We consider recent work on external-memory graph search, including duplicate-detection strategies (delayed, hash-based, and structured); integration of these strategies in external-memory versions of breadth-first search, breadth-first heuristic, and frontier search; the inclusion of a perfect hash function, as well as combining parallel and disk-based search; external-memory pattern-database heuristics; and applications of external-memory search to AI planning, automated verification, and other search problems. Implicit graph search that is in the scope of the seminar includes deterministic and non-deterministic models, as well as game-theoretical and probabilistic models.

Moreover, the seminar is specifically concerned with algorithm designs for implicit graph search on modern personal computer architectures, e.g. subject to several processing units on the graphic card, and hierarchical memory including solid state disks. Applications areas for new algorithm designs that exploit modern hardware are found in the model checking community, but also in AI planning and game-playing.

The industrial impact is located mostly in the area of software validation, and to some extent in the area of hardware verification.

Computational models for flash memory devices

Deepak Ajwani (Aarhus University, DK)

Based on the characteristics of the flash devices [1], we propose two new computation models [2] – the general flash model and the unit-cost flash model. The general flash model is similar to the I/O model, with the exception that read and write block sizes are different and that they incur different costs. The unit-cost flash model augments the general flash model with the assumption that the throughput provided by sequential reads and writes is equal.

These models are generic (as they abstract away many device-specific characteristics) and simple enough for meaningful algorithm design. In particular, we show that a large body of existing external memory algorithms and data structures based on the merging paradigm can be easily adapted to give efficient algorithms in the unit-cost flash model.

References

1. Ajwani, Deepak; Malinger, Itay; Meyer, Ulrich; Toledo, Sivan; *Characterizing the Performance of Flash Memory Storage Devices and Its Impact on Algorithm Design*, Workshop on Experimental and Efficient Algorithms (WEA), 2008
2. Ajwani, Deepak; Beckmann, Andreas; Jacob, Riko; Meyer, Ulrich; Moruz, Gabriel; *On Computational Models for Flash Memory Devices*, 8th International Symposium on Experimental Algorithms (SEA), 2009

Joint work of: Ajwani, Deepak; Beckmann, Andreas; Jacob, Riko; Meyer, Ulrich; Moruz, Gabriel

See also: Deepak Ajwani, Andreas Beckmann, Riko Jacob, Ulrich Meyer and Gabriel Moruz. On Computational Models for Flash Memory Devices. 8th International Symposium on Experimental Algorithms. 2009

K*: A Heuristic Search Algorithm for Finding the k Shortest Paths

Husain Aljazzar (Universität Konstanz, DE)

We present a new directed algorithm, called K*, for finding the k shortest paths between a designated pair of vertices in a given directed weighted graph. As a directed algorithm, K* has two advantages compared to current k-shortest-paths algorithms. First, K* performs on-the-fly, which means that it does not require the graph to be explicitly available and stored in main memory. Portions of the graph will be generated as needed. Second, K* can be guided using heuristic functions. This leads to significant improvements in the memory and runtime demands for many practical problem instances. We prove the correctness of K* and determine its asymptotic worst-case complexity as $O(m + n \log n + k)$ with respect to both runtime and space, where n is the number of vertices and m is the number of edges of the graph. We performed some experiments using two case studies from two different application domains namely route planning and counterexample generation for stochastic model checking. The experimental results illustrate the efficiency and scalability of K* compared to most efficient k-shortest-paths algorithms known so far.

Key words: K*, heuristic search, k-shortest-paths problem

References

1. D. Eppstein,*Finding the k shortest paths*, SIAM J. Computing 28 (2), 1998
2. H. Aljazzar, S. Leue, *Directed explicit state-space search in the generation of counterexamples for stochastic model checking*, IEEE Trans. Softw. Eng., 2009
3. H. Aljazzar, *Directed diagnostics of system dependability models*, Ph.D. dissertation, University of Konstanz, 2009, URL: *http://kops.ub.uni-konstanz.de/volltexte/2009/9188/*
4. H. Aljazzar, S. Leue, *Generation of counterexamples for model checking of markov decision processes*, 6th International Conference on the Quantitative Evaluation of SysTems (QEST '09), IEEE Computer Society Press, 2009.

Massive-scale graph analytics

David A. Bader (Georgia Institute of Technology, US)

Graph abstractions are proving useful in solving real-world challenges in traditional and emerging computational sciences such as chemistry, biology, and medicine, as well as applications in national security. A well-known centrality metric, betweenness, has been used in the study of lethality in protein interaction networks [1], the spread of disease through human contact [2], identifying key actors in terrorist networks [3], and transportation networks [4]. Interestingly, graphs originating from a number of unrelated disciplines, including computer networks, biological networks, and social networks, exhibit common structural characteristics such as a low diameter and a power-law distribution in the number of neighbors [9]. These same topological properties increase the difficulty of obtaining balanced graph partitioning and performing fast analysis on distributed memory, cache-based supercomputers.

The explosion of real-world graph data poses a substantial challenge: How can we analyze constantly changing graphs with billions of vertices? Our approach uses multithreaded high-performance computers such as the Cray XMT to scale to massive graphs. These machines utilize thousands of hardware threads to tolerate the latency of memory accesses inherent in massive graph analysis. We have demonstrated the Cray XMT's ability to analyze graphs several orders of magnitude larger than standard social network analysis packages using algorithms such as betweenness centrality [10,11] and clustering coefficients [12]. A new variation on a traditional analysis metric, k-betweenness centrality is more resilient to the noisy data with false positives and false negatives inherent in massive social networks and scales up to billions of vertices on a 128 processor XMT [13]. We have studied compact graph representations for spatio-temporal data [14] and have offered an extensible data structure for dynamic graph data that is suitable for a wide variety of applications with little or no performance penalty when compared to static representations [8].

In this talk, we will discuss several important problems in real-world graphs. We will look at the use of centrality metrics, such as betweenness [7], to identify important vertices in a graph, as well as techniques for understanding community structure. These include algorithms for identifying communities [6] as well as key metrics for characterizing a graph [5].

References

1. Jeong, H., Mason, S.P., Barabási, A.-L., Oltvai, Z.N., *Lethality and centrality in protein networks*, Nature, 412, pages 41–42, 2001
2. Liljeros, F., Edling, C.R., Amaral, L.A.N., Stanley, H.E., Åberg, Y., *The Web of Human Sexual Contacts*, Nature, 411, pages 907–908, 2001
3. Coffman, T., Greenblatt, S., Marcus, S., *Graph-based technologies for intelligence analysis*, Commun. ACM, 47, pages 45–47, 2004

4. Guimerà, R., Mossa, S., Turtschi, A., Amaral, L.A.N., *The worldwide air transportation network: Anomalous centrality, community structure, and cities' globalroles*, Proc. National Academy of Sciences USA, 102, 22, pages 7794–7799, 2005
5. Watts, D.J., Strogatz, S.H., *Collective dynamics of small world networks*, Nature , 393, pages 440-442, 1998
6. Girvan, M., Newman, M.E.J., *Community structure in social and biological networks*, Proc. National Academy of Sciences USA, 99, 12, pages 7821–7826, 2002
7. Freeman, L.C., *A set of measures of centrality based on betweenness*, Sociometry, 40, 1, pages 35–41, 1977
8. Bader, D.A., Berry, J., Amos-Binks, Chavarría-Miranda, D., Hastings, C., Madd, K., *STINGER: Spatio-Temporal Interaction Networks and Graphs (STING) Extensible Representation*, Georgia Institute of Technology, 2009
9. Newman, M.E.J., *The structure and function of complex networks*, SIAM Review, 45, 2, pages 167–256, 2003
10. Bader, D.A., Madduri, K., *Parallel Algorithms for Evaluating Centrality Indices in Real-world Networks*, Proc. 35th Int'l Conf. on Parallel Processing (ICPP), 2006
11. Madduri, K.Ediger, D., Jiang, K., Bader, D.A., Chavarría-Miranda, D., *A Faster Parallel Algorithm and Efficient Multithreaded Implementations for Evaluating Betweenness Centrality on Massive Datasets*, Lawrence Berkeley National Laboratory, 2009
12. Ediger, D., Jiang, K., Riedy, J., Bader, D.A., *Massive Streaming Data Analytics: A Case Study with Clustering Coefficients*, Proc. Workshop on Multithreaded Architectures and Applications (MTAAP'10), 2010
13. Jiang, K., Ediger, D., Bader, D.A., *Generalizing k-Betweenness Centrality Using Short Paths and a Parallel Multithreaded Implementation*, Proc. 38th Int'l Conf. on Parallel Processing (ICPP), 2009
14. Madduri, K., Bader, D.A. *Compact Graph Representations and Parallel Connectivity Algorithms for Massive Dynamic Network Analysis*, 23rd IEEE International Parallel and Distributed, 2009

Keywords: Parallel algorithms, high-performance computing, social network analysis

Challenges and perspectives of multi-core and GPU model checking

Dragan Bosnacki (TU Eindhoven, NL)

Model checking is one of the most successful formal techniques for the verification of software and hardware systems. Developed in the beginning of the eighties, nowadays it is used by major companies, like Microsoft, to improve the quality of their products. So called explicit state model checking boils down to exhaustive search of the state space of the system which is verified. The state space can be seen as a labeled graph and therefore many of the model checking algorithms are essentially graph search algorithms. Since those graphs are usually huge, for practical applications of model checking it is essential to reduce as much as possible the time and memory requirements of the algorithms.

In my talk I focus on runtime improvements in model checking by exploiting the recent developments in the processor technology. In particular, the emphasis will be on parallel algorithms for shared memory architectures, like multi-core systems and general purpose graphic processing units (GPGPU). These research directions are quite new - the first papers on multi-core and GPU model checking were published in 2007 and 2009, respectively [2,1]. Therefore, there is still a lot of uncharted territory which could be a fruitful ground for cooperation of experts from different computer science communities.

After presenting a brief history of the multi-core and GPU model checking, I make an inventory of challenges and open problems in these new research avenues. For instance, one of the main not yet achieved goals is to find an efficient (shared memory) parallel algorithm for finding cycles in the state space graph. As a topic this could be interesting also for the seminar participants who do not work directly in model checking. For the people who are involved in model checking there are challenges like compatibility of the new algorithms with the state space reduction techniques, e.g. partial-order reduction. Such techniques are the main weapon in model checking to cope with huge state spaces.

Also, I discuss some concrete ideas and directions for solutions to some of these problems.

I wrap up with some avenues for future work within both multi-core and GPU model checking.

References

1. D. Bošnački, S. Edelkamp, D. Sulewski, *Efficient Probabilistic Model Checking on General Purpose Graphics Processors*, Model Checking Software, 16th International SPIN Workshop, Lecture Notes in Computer Science 5578, pp. 32-49, Springer, 2009.
2. G.J. Holzmann, D. Bošnački, *The Design of a Multi-core Extension of the SPIN Model Checker*, IEEE Trans. on Software Engineering, Vol. 33, No. 10, pp. 659-674, October 2007.

Keywords: Formal verification, model checking, multi-core model checking, gpu model checking

Roomy: A New Approach to Parallel Disk-based Computation

Gene Cooperman (Northeastern Univ. - Boston, US)

Graph Search often runs into problems requiring large storage.

When such computations overrun available RAM, the traditional workarounds have been:

1. large-memory shared memory computers (expensive)
2. use of aggregate RAM of a computer cluster (needs parallel programming)
3. use of disk on a single computer (slower than RAM)

We discuss a fourth paradigm, disk-based parallel computing, and a new programming language, Roomy, for this new paradigm. The aggregate bandwidth of the disks of a local cluster (or of a SAN) is comparable to the bandwidth of a *single* RAM subsystem. Hence, our goal is to replace large-memory *sequential* programs by disk-based parallel programs.

The goal is not greater speed, but greater storage.

Prior to Roomy, our working group had several successes in computational group theory, and in upper bounds on the number of moves needed to solve Rubik's Cube. Each of these successes was accompanied by months of programming pain. What do you do when on the third day of the computation, on the 23rd computer node, the second thread of the application process discovers an error in the 537th file?

Roomy provides an easy-to-use mini-language (in fact, an API and run-time library for C/C++) which emphasizes a streaming-compatible style of programming. This does not eliminate latency issues, but the Roomy language naturally biases users toward avoiding latency.

The Roomy software is available at: http://roomy.sourceforge.net

Keywords: Disk-based parallel computation, external memory

Joint work of: Cooperman, Gene; Kunkle, Daniel

Hash, Displace, and Compress

Martin Dietzfelbinger (TU Ilmenau, DE)

A hash function h, i.e., a function from the set U of all keys to the range $[m] = \{0, \ldots, m-1\}$ is called a perfect hash function (PHF) for a subset $S \subseteq U$ of size $n \leq m$ if h is 1–1 on S. It is called a minimal perfect hash function (MPHF) if in addition m is as small as possible, i.e., $m = n$. The important performance parameters of a PHF are representation size (measured in bits per key), evaluation time (which should be worst case constant) and construction time (which should be expected linear).

In [1] a construction was presented that requires $m = 1.23n$, expected linear construction time and storage space 1.97 bits per key. From this, using known techniques for succinct data structures (see, e.g., [6]) one can obtain a construction of a MPHF with 2.61 bits per key. The central idea in this construction is to use 3 fully random hash functions, and utilize the fact that if $m > 1.23n$ the probability is high that the 3-uniform hypergraph created by the n sets of hash values for each key is acyclic.

In this contribution, we present an algorithm that makes it possible to obtain PHFs and MPHFs with even less storage space. The expected representation size is not far from optimal while the expected construction time remains at $O(n)$ and the evaluation time remains $O(1)$ in the worst case. In theory, there is a tradeoff between construction time and space for the hash function. But especially in experiments, our scheme works nicely and better than previous schemes. For example in the case $m = 1.23n$ we obtain a PHF that uses space 1.4 bits per key (the information theory lower bound is 0.88 bits per key). For $m = 1.01n$ ("almost perfect") we obtain space 1.98 bits per key, which was not achievable with previously known methods. Using succinct data structure techniques, we obtain MPHFs with 2.07 bits per key.

Our algorithm is inspired by several known algorithms; the main new feature is that we combine a modification of Pagh's [5] "hash-and-displace" approach with data compression on a sequence of hash function indices, see e.g. [4,3]. That combination makes it possible to significantly reduce space usage while retaining linear construction time and constant query time.

For the analysis we assume that fully random hash functions are given for free; such assumptions can be justified and were made in previous papers. Our algorithm can also be used for k-perfect hashing, where at most k keys may be mapped to the same value.

In the context of graph search, perfect hash functions are useful for example for semi-external searching algorithms, keeping in internal memory a flag array with only one bit of information for each key, as demonstrated in the paper [2] by Edelkamp, Sanders, and Šimeček, also in this seminar. A serious limitation is that the graph has to be traversed once first to identify all nodes. So for this and other applications it would be interesting to study (partly) dynamic versions of the algorithm.

References

1. Botelho, Fabiano C., Pagh, Rasmus, Ziviani, Nivio, *Simple and space-efficient minimal perfect hash functions*, 10th International Workshop on Algorithms and Data Structures (WADS), 2007, pp. 139–150.
2. Edelkamp, Stefan, Sanders, Peter, v Simeček, Pavel, *Semi-external LTL Model Checking*, 20th International Conference on Computer Aided Verification (CAV), 2008, pp. 530–542
3. Ferragina, Paolo, Venturini, Rossano, *A Simple Storage Scheme for Strings Achieving Entropy Bounds*, Theoretical Computer Science **372**(1), 2007, 115–121.
4. Fredriksson, Kimmo, Nikitin, Fedor, *Simple compression code supporting random access and fast string matching*, 6th International Workshop on Experimental Algorithms (WEA), 2007, pp. 203–216.
5. Pagh, Rasmus, *Hash and displace: Efficient Evaluation of Minimal Perfect Hash Functions*, 6th International Workshop on Algorithms and Data Structures (WADS) 1999, pp. 49–54.
6. Vigna, Sebastiano, *Broadword Implementation of Rank/Select Queries*, 7th International Workshop on Efficient and Experimental Algorithms (WEA), pp. 154–168.

Keywords: Hashing, Data Structures, Dictionaries, Random Graphs, Data Compression, Semi-External Graph Search

Joint work of: Belazzougui, Djamal; Botelho, Fabiano C.; Dietzfelbinger, Martin

See also: Belazzougui, Djamal, Botelho, Fabiano C., Dietzfelbinger, Martin, *Hash, Displace, and Compress*, 17th Annual European Symposium on Algorithms (ESA), 2009, Springer LNCS 5757, pp. 682–693.

Rank-Relaxed Weak Queues

Stefan Edelkamp (Universität Bremen, DE)

A run-relaxed weak queue by Elmasry et al. [2] is a priority queue data structure with insert and decrease-key in $O(1)$ as well as delete and delete-min in $O(\log n)$ worst-case time. One further advantage is the small space consumption of $3n + O(\log n)$ pointers.

In this paper we propose rank-relaxed weak queues, reducing the number of rank violations nodes for each level to a constant, while providing amortized constant time for decrease-key. Compared to run- relaxed weak queues, the new structure additionally gains one pointer per node.

An empirical evaluation shows that the implementation can out- perform Fibonacci and pairing heaps in practice even on rather simple data types.

References

1. Edelkamp, S., *Rank-Relaxed Weak Queues: Faster than Pairing and Fibonacci Heaps?*, Technical Report 54, TZI Universität Bremen, 2009
2. A. Elmasry, C. Jensen, and J. Katajainen. *Relaxed weak queues: An alternative to run-relaxed heaps.*, Technical Report CPH STL 2005-2, Department of Computing, University of Copenhagen, 2005.
3. J. Rasmussen, *Implementing run-relaxed weak queues.*, Technical Report CPH STL 2008-1, Department of Computing, University of Copenhagen, 2008.

Keywords: Priority Queue, Algorithm Engineering

Joint work of: Edelkamp, Stefan; Katajainen, Jyrki; Rasmussen, Jens

Leveraging Local Structure in External-Memory Graph Search

Eric Hansen (Mississippi State University, US)

This talk provides an overview of recent work on how to leverage graph structure to improve the efficiency of external-memory breadth-first search.

I begin with a review of approaches to leveraging "interlayer locality" in breadth-first search, where interlayer locality means that duplicate nodes are more likely in adjacent layers of the graph. The focus of the talk, however, is on approaches to leveraging "intralayer locality," which means that duplicates tend to occur near each other in the sequence of nodes generated when expanding the nodes within a layer. I point out that interlayer locality depends on the order in which the nodes in a breadth-first layer are expanded, and then review two approaches to external-memory breadth-first search that leverage this form of local structure.

I begin with a review of structured duplicate detection, an approach to external-memory graph search that uses an abstraction of the state space to localize references to stored nodes, allowing immediate duplicate detection. After pointing out that structured duplicate detection can be viewed as a kind of perfect cache, I describe an alternative approach to leveraging intralayer locality that uses an LRU hash table in RAM to detect most duplicates immediately, relying on delayed duplicate detection as a backup. The search strategies described in this talk have been applied to both artificial intelligence planning and model checking using the Murphi verifier, and preliminary results are reported.

Keywords: External-memory graph search, breadth-first search, model checking

Joint work of: Hansen, Eric; Lamborn, Peter

Landmarks, Critical Paths and Abstractions: What's the Difference Anyway?

Malte Helmert (Universität Freiburg, DE)

Current heuristic estimators for classical domain-independent planning are usually based on one of four ideas: delete relaxation, abstraction, critical paths, and, most recently, landmarks.

Previously, these different ideas for deriving heuristic functions were largely unconnected.

In my talk, I will show that these heuristics are in fact very closely related. Moreover, I will introduce a new admissible heuristic called the landmark cut heuristic which exploits this relationship. In our experiments, the landmark cut heuristic provides better estimates than other current admissible planning heuristics, especially on large problem instances.

Keywords: Planning, heuristic search, heuristic functions

Joint work of: Helmert, Malte; Domshlak, Carmel

Extended Abstract: http://drops.dagstuhl.de/opus/volltexte/2010/2432

Full Paper:
http://www.informatik.uni-freiburg.de/~ki/papers/helmert-domshlak-icaps2009.pdf

See also: In Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009), pp. 162-169. AAAI Press 2009.

Sparse Matrix Multiplication in the I/O-Model

Riko Jacob (TU München, DE)

Sparse matrix multiplication is a generalization of permuting, one of the fundamental problems that are optimally solved in the I/O-model using sorting (Aggarwal/Vitter CACM 1988).

Hence, it is no surprise that also for this problem a sorting based approach is optimal (Bender/Brodal/Fagerberg/Jacob/Vicari SPAA 2007), and this result actually also covers the situation of map-reduce.

Further, there are some recently published results and ongoing work on parallel external memory models, (simultaneous) multiplication of several vectors, and Sparse Matrix - Dense Matrix multiplication (Greiner/Jacob LATIN 2010).

BDDs in Planning and General Game Playing

Peter Kissmann (TU Dortmund, DE)

In this talk we will study symbolic algorithms to solve both planning problems and general single- and two-player games.

For the planning domain we look at sequential optimal planning with action costs with and without soft constraints. For the first case we use either symbolic (bidirectional) BFS in case of no action costs, so that we come up with step-optimal plans, or symbolic versions of Dijkstra's algorithm and A* in case of non-uniform action costs, resulting in cost-optimal plans. In the second case, we came up with an extension of a symbolic Branch-and-Bound algorithm, which then returns plans achieving an optimal net-benefit (rewards for achieving weak goals minus total action-costs) [3].

General game playing might be seen as a generalization of planning: Where planning typically concerns only one player, general game playing copes with single-player as well as multi-player games. Here we will present symbolic algorithms that can solve general single-player games [1] as well as two-player games [2] – with the only restriction of the games being turn-taking.

References

1. Edelkamp, Stefan, Kissmann, Peter, *Symbolic Exploration for General Game Playing in PDDL*, ICAPS-Workshop on Planning in Games, 2007.
2. Edelkamp, Stefan, Kissmann, Peter, *Symbolic Classification of General Two-Player Games*, 31st Annual German Conference on Artificial Intelligence (KI), volume 5243 of LNCS, pp. 185–192, Springer, 2008.

3. Edelkamp, Stefan, Kissmann, Peter, *Optimal Symbolic Planning with Action Costs and Preferences*, 21st International Joint Conference on Artificial Intelligence (IJ-CAI), pp. 1690–1695, 2009.

Keywords: Symbolic Search, Planning, General Game Playing

Joint work of: Kissmann, Peter; Edelkamp, Stefan

Specification and Analysis with Hierarchical Graphs

Alberto Lluch Lafuente (1) IMT Institute for Advanced Studies Lucca 2) Computer Science Department, University of Pisa)

Architectural Design Rewriting [1,4] conciliates algebraic and graph-based techniques to offer a flexible model for software systems (e.g. software architectures) and their dynamics (e.g. architectural reconfiguration). Roughly, ADR models are rewrite theories interpreted over a particular class of hierarchical graphs. The algebraic presentation of graphs [2] enables the use standard term rewriting techniques to model system dynamics, while the graphical representation allows us to enjoy intuitive visual notations.

The spirit of our work is to characterise classes of software systems with many-sorted algebras, where each sort stands for a class of systems having a certain structure, and whose operations are well-typed compositions, i.e. the way well-typed systems are composed into a well-typed system. In this view, each system can be described by a well-typed term of the algebra, which denotes how the system is composed and how it can be decomposed. Moreover, because we can equip the algebra with structural axioms, we are actually dealing with equivalence classes of terms which, means that for each system we are actually given a set of possible decompositions that can be exploited to declare the dynamics of a system as rewrite rules. Moreover, terms are interpreted as graphs and the term equivalence amounts to graph isomorphism. This offers an intuitive graphical representation of systems, since equivalent terms are represented by isomorphic graphs. The flexibility of the approach has been validated over a variety of heterogeneous software system topologies and their dynamics. For instance, algebraic presentations and rewrite rules upon them can be used to define meta-models and model transformations, software architectures and architectural reconfiguration, network topologies and network reconfiguration or graphical presentations of states and semantics of process algebras. Besides the strong mathematical foundation, our approach can be mapped into Rewriting Logic, a kind of conditional rewrite systems enjoying an efficient implementation in the Maude rewrite engine, which provides built-in tools such as a model checker and a theorem prover [3].

Our current research efforts are devoted to the development of appropriate property specification mechanisms and efficient analysis techniques. For instance, one of the crucial inherent problems of the dynamics of the systems we are interested in is due to the non-deterministic nature of rewrite rules, which gives rise to considerably large state spaces. In practice, we have to deal with large spaces of possible software architectures, network configurations, software models, or process terms. But in most cases we are just interested in finding a good one according to some quantitative criteria. For instance, network reconfigurations ensuring a good load-balance but involving a low number of network disconnections, class diagram refactorings reducing the number of classes but not requiring too many method pull-ups, or executions in a session-oriented calculus involving few steps but resulting in high nesting depths.

Therefore one of the main problems we are interested in is a sort of planning problem, which, however, have some idiosyncrasies that makes them different from traditional approaches like *action planning*. First, states in traditional planning tend to be *flat*, i.e. they typically consists of sets of ground predicates. Instead, our states are *structured*, i.e. they are terms and hence we are dealing with a more general notion of a state. Second, rules in traditional planning (i.e. actions) are first order and unconditional. Instead, our rules are conditional term rewrite rules, i.e. variables can stand for subterms and conditions can be rewrite rules whose result we can use in the right-hand side. Moreover, our rules are typically equipped with labels that are used to coordinate and guide the rewriting over the structure. Technically, our rules are in the style of Structural Operational Semantics. Last, we are interested in multi-criteria optimisation, i.e. we would like to consider several dimensions to optimise and find non-dominated optimal or near-to-optimal solutions. The criteria should be modelled with an algebraic structure that decouples the planning algorithm from the particular criteria instance. For instance, one possible choice are semi-rings, the mathematical cost structure underlying Floyd-Warshall's all-pairs shortest paths algorithm.

Such planning problems can be modelled with rewrite systems in a declarative way, for instance using terms to describe sets of states associated with their quantitative attributes, equations to discard dominated states and a simple rule to expand the set of states by applying a planning step. Then, the ordinary exploration mechanisms of rewrite systems can be solved to explore this state space of state sets. Declarative approaches approaches are usually very inefficient. We believe that there is space for the application of existing algorithms and techniques and, possibly, for the development of new ones.

Acknowledgements. Joint work with Roberto Bruni, Ugo Montanari and Generoso Paolillo within the EU FET integrated projects SENSORIA and the Italian FIRB Project TOCAI.IT.

References

1. ADR homepage. http://www.albertolluch.com/adr.html.
2. R. Bruni, F. Gadducci, and A. Lluch Lafuente. A graph syntax for processes and services. In S. Jianwen and C. Laneve, editors, *WS-FM'09*, LNCS. Springer Verlag, 2009. To Appear.

3. R. Bruni, A. Lluch Lafuente, and U. Montanari. Hierarchical design rewriting with maude. In G. Rosu, editor, *WRLA 2008*, volume 238 (3) of *Electronic Notes in Theoretical Computer Science*, pages 45–62. Elsevier, 2009.
4. R. Bruni, A. Lluch Lafuente, U. Montanari, and E. Tuosto. Style Based Architectural Reconfigurations. *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, 94:161–180, February 2008.

Keywords: Graphs, term rewriting, process calculi, graph logics, planning, software architectures

On parallel UCT

Hartmut Messerschmidt (Universität Bremen, DE)

In game theory there are often very large game trees. It turned out, that Monte Carlo Tree Search is an appropriate method for searching in these trees. Almost all good GO programs and many programs in the general game playing community are using UCT (Upper Confidence bounds applied on Trees).

UCT combines Monte Carlo Tree Runs (MCTR) with Upper Confidence Bounds (UCB, see [1]), a method to get the highest reward during a series of games. In UCB an evaluation function is used to determine the next game to play. This function depends on the number of plays and the current known values of the games. There is a balance between exploration, that is exploring other games than the best to see if they can do better, and exploitation (choose the best known game). In UCT this method is used to construct a game tree and in each step the evaluation function is used to determine which child to explore further. Monte Carlo Tree Runs are used to determine the value of a leaf.

The UCT algorithm was first introduced by Kosics and Szepesvari (see [**?**]).

Each run starts from the root and chooses one child to expand next. This is done by the evaluation function that depends on the values and the counters of the children. If it reaches a leaf then a MCTR starts to determine the value of that leaf. In the second phase this value is propagated back to the root. A node stores for example the average or the maximum value. During this process the counters of all visited nodes are increased by one, too.

As UCT turned out to be a good tool to explore game trees, and it becomes even better with more computational power, it is natural to look for a way to parallelize UCT. UCT is a sequential algorithm, so more precisely the question is if a parallel algorithm exists that is somehow close to UCT?

The problem in parallelizing UCT is that if many threads run in a single UCT tree, then, in the original UCT, all will expand the same nodes. This is the case, because the move from the root to a leaf of the game tree is deterministic.

Cazenave and Jouandeau ([3] and [4]) presented three different approaches to do parallelization for UCT. But they either work on different game trees, or use one master to move through the game tree and do the MCTRs in parallel.

Although these approaches lead to good results, the question is whether the UCT algorithm can get a speed up from parallelization without altering it too much.

Another approach comes from Enzenberger and Müller ([5]). They use one game tree for all threads to improve the Fuego Go program. Unfortunately it is not explained how the threads are controlled to move in different paths.

The main idea here is to split the update process. While moving down to a leaf the counters are increased and on the way back only the values are changed. This ensures that different threads take different paths through the game tree.

As this leads to a speed up of the algorithm, the next question is how to overcome the space limitations by freeing memory that belongs to unimportant parts of the game tree. A part of the game tree becomes unimportant, when it is completely explored, or it leads to a sure win or loss for one side.

References

1. Auer, Peter, *Using Confidence Bounds for Exploitation-Exploration Trade-offs*, Journal of Machine Learning Research 3, pages 397-422, 2002
2. Kocsis, L. and Szepesvari, C., *Bandit based Monte-Carlo planning*, In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp 282-293.Springer, 2006
3. Cazenave, T. and Jouandeau, N., *On the parallelization of UCT.*, In Computer Games Workshop, pages 93-101, Amsterdam, 2007
4. Cazenave, T. and Jouandeau, N., *A parallel Monte-Carlo tree search algorithm.*, In J. van den Herik, X. Xu, Z. Ma, and M. Winands (eds.), Computer and Games, vol. 5131 LNCS, pages 72-80, Springer, 2008
5. Enzenberger, M. and Müller, M., *A lock-free multithreaded Monte-Carlo tree search algorithm*, Accepted for ACG12, 2009

Keywords: UCT, parallel, games, morpion solitaire

Via Detours to I/O-Efficient Shortest Paths

Ulrich Carsten Meyer (J.W. Goethe Universität Frankfurt, DE)

Large explicit graphs arise naturally in many real world applications. The actual performance of simple RAM model algorithms for traversing these graphs (stored as adjacency-lists in external memory) deviates significantly from their linear or near-linear predicted performance because of the large number of I/Os they incur. In order to alleviate the I/O bottleneck, many external memory graph traversal algorithms have been designed with provable worst-case guarantees.

Based on [1] we review some detours, dead-ends, and happy ends of our still ongoing research on external-memory shortest-path algorithms.

References

1. Meyer, Ulrich, *Via Detours to I/O-Efficient Shortest Paths*, Efficient Algorithms, volume 5760 of LNCS, pp. 219-232, Springer, 2009.

Keywords: External Memory, Graph Algorithms

Full Paper:
http://www.springerlink.com/content/v78603330256m540/

GPU Sample Sort

Vitaly Osipov (KIT - Karlsruhe Institute of Technology, DE)

We present the design of a sample sort algorithm for manycore GPUs. Despite being one of the most efficient comparison-based sorting algorithms for distributed memory architectures its performance on GPUs was previously unknown. For uniformly distributed keys our sample sort is at least 25% and on average 68% faster than the best comparison-based sorting algorithm [1], GPU Thrust merge sort, and on average more than 2 times faster than GPU quicksort [2]. Moreover, for 64-bit integer keys it is at least 63% and on average 2 times faster than the highly optimized GPU Thrust radix sort [1] that directly manipulates the binary representation of keys. Our implementation is robust to different distributions and entropy levels of keys and scales almost linearly with the input size. These results indicate that multi-way techniques in general and sample sort in particular achieve substantially better performance than two-way merge sort and quicksort.

References

1. Satish, Nadathur, Harris, Mark, Garland, Michael *Designing Efficient Sorting Algorithms for Manycore GPUs*, Proc. Int'l Symposium on Parallel and Distributed Processing (IPDPS), 2009
2. Cederman, Daniel, Tsigas, Philippas *A Practical Quicksort Algorithm for Graphics Processors*, Proc. European Symposium on Algorithms (ESA), 2008

Keywords: Sorting, GPU, manycore, multicore

Joint work of: Leischner, Nikolaj; Osipov, Vitaly; Sanders, Peter

Graph Search Engineering in Formal Verification

Kairong Qian (Synopsys Inc. - Mountain View)

Many formal verification problems involve searching the graph for some goal state(s). We present an industrial perspective of how graph search algorithms are deployed to solve formal verification problems. In particular, we discuss the implicit graph search in model checking using a broad range of techniques. For reachability search, we emphasize the blend of BDDs and Boolean SAT methods to achieve both DFS and BFS. Finally, we also talk about how commercial formal tools use the orchestration to utilize the strength of each individual technology. We conclude by remarking the important research issues that still remains open from a commercial stand point.

Best-first Graph Search on Multicore Machines

Wheeler Ruml (Univ. of New Hampshire - Durham, US)

Best-first graph search algorithms such as Dijkstra's algorithm and A* are of fundamental importance. I'll present recent work with Ethan Burns, Seth Lemons, and Rong Zhou on adapting best-first graph search to shared-memory parallel computers. Previous proposals for parallel best-first graph search either abandon a best-first search order or fail to exploit parallelism efficiently. We've come up with a simple approach that, in our experiments on a dual quad-core Intel machine, seems quite effective [1].

It is based on dividing work using an abstraction of the search space, as in structured duplicate detection [3]. Each thread acquires a portion of the search space that allows it to expand nodes and check for duplicates without requiring further syncronization. A heap holds those portions of the search space that are available for searching, sorted by the f value of their best node. Threads greedily acquire the most promising portion to expand. A simple implementation of this scheme leads to livelock, but we have proved that a simple modification suffices to avoid it.

The simplicity of our approach allows it to easily handle domains with non-uniform or real-valued edge costs [2]. In addition, it extends naturally to bounded-suboptimal search (in analogy to weighted A*) and anytime search (in analogy to Anytime Weighted A*).

In this work, we assume a shared-memory machine, but our method should combine well with other proposals for distributed search. It is an interesting topic for future work to extend this method to use external memory.

References

1. Burns, Ethan; Lemons, Seth; Zhou, Rong; and Ruml, Wheeler, *Best-first heuristic search for multi-core machines*, Proceedings of IJCAI, 2009.

2. Burns, Ethan; Lemons, Seth; Ruml, Wheeler; and Zhou, Rong, *Suboptimal and Anytime Heuristic Search on Multi-Core Machines*, Proceedings of ICAPS, 2009.
3. Zhou, Rong; and Hansen, Eric A., *Parallel structured duplicate detection*, Proceedings of AAAI, 2007.

Keywords: Combinatorial search, heuristic search, parallel algorithms

Full Paper:
http://www.cs.unh.edu/~ruml/papers/pbnf-socs-09.pdf

Exploiting the Locality of Transitions

Theo Ruys (University of Twente, NL)

We are concerned with the generation of states for concurrent models. More specifically, the state space generation of explicit state, software model checkers. Explicit state model checkers are controlled by interleaving semantics. That is, the execution of processes is interleaved: statements of different processes do not occur at the same time.

Due to this interleaving semantics, a global transition of the system is typically only local: only one process performs an execution step. Most algorithms in a model checker (e.g., SPIN), however, are global in nature and operate on the complete representation of a state. For example: checking whether a state has already been seen [1], algorithms for state compression, algorithms for storing states, garbage collection algorithms [2], etc.

In this talk we focus on several well-known global algorithms and show how these algorithms can be turned into incremental ones. Hence, we propose the exploitation of local transitions within a model checker. This could lead to improvements in both space and time. This is especially the case for large software models [3] which have huge state descriptors.

References

1. Viet Yen Nguyen and Theo C. Ruys, *Incremental Hashing for Spin*, Proc. of SPIN 2008, LNCS 5156, pp. 232–249, 2008.
2. Viet Yen Nguyen and Theo C. Ruys, *Memoised Garbage Collection for Software Model Checking*, Proc. of TACAS 2009, LNCS 5505, pp. 201–214, 2009.
3. Niels H. M. Aan de Brugh, Viet Yen Nguyen and Theo C. Ruys, MoonWalker*: Verification of .NET Programs*, Proc. of TACAS 2009, LNCS 5505, pp. 170–173, 2009.

Keywords: Explicit state, model checking, transitions, locality

Engineering Route Planning Algorithms – Current Activities

Peter Sanders (KIT - Karlsruhe Institute of Technology, DE)

Computing an optimal route in a transportation network between specified source and target nodes is one of the showpieces of real-world applications of algorithmics. We frequently use this functionality when planning trips with cars or public transportation.

There are also many applications like logistic planning or traffic simulation that need to solve a huge number of shortest-path queries in transportation networks.

In the last decade, new methods have been developed, that dramatically speed up search in such situations without sacrificing accuracy. These methods vary with their appraoch: hierarchical, goal directed or based on distance tables. Also, combinations make a lot of sense [3]. In particular, the hierarchy implicit in road networks has helped a lot here. Depending on the preprocessing time and space invested, these methods allow speedups of up to six orders of magnitude compared to Dijkstra's algorithm on continental sized road networks [1]. A recent overview paper [5] explains many of these methods.

In the last few years, research has focused on advanced models where the graph changes dynamically, e.g., due to traffic jams [8], where edge weights depend on the time of travel [4,2], when there are multiple objective functions [6], or when the route planning is done on a mobile device [7].

Major open problems are the extension of the most successful methods to other graph families and more theoretical insights why and when speedup techniques work well. We also need a more realistic treatment of traffic jams. For example, one could combine information about traffic jams with real time traffic simulation in order to find out where secondary traffic jams are likely.

Acknowledgements

We would like to thank our coauthors on route planning, Hannah Bast, G. Veit Batz, Reinhard Bauer, Daniel Delling, Stefan Funke, Robert Geisberger, Sebastian Knopp, Moritz Kobitzsch, Domagoj Matijevic, Jens Maue, Dennis Schieferdecker, Dominik Schultes, Christian Vetter and Lars Volker for their valuable contributions. We also had many interesting discussions with Rob van den Berg, Sebastian Egner, Andrew Goldberg, Ulrich Lauther, Ramon Lentink, Dennis Luxen, Rolf Möhring, Paul Perdon, Martin Pfeifle, Heiko Schilling, Frank Schulz, Mikkel Thorup, Thomas Willhalm, Jaques Verrier, Peter Vortisch, and Renato Werneck. Finally, we thank PTV AG and HaCon for providing us with real-world data for scientific use.

References

1. H. Bast, S. Funke, P. Sanders, and D. Schultes. Fast routing in road networks with transit nodes. *Science*, 316(5824):566, 2007.

2. G. Batz, D. Delling, P. Sanders, and C. Vetter. Time-dependent contraction hierarchies. In *10th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 97–105, 2009. http://algo2.iti.uni-karlsruhe.de/1222.php.
3. R. Bauer, D. Delling, P. Sanders, D. Schieferdecker, D. Schultes, and D. Wagner. Combining hierarchical and goal-directed speed-up techniques for Dijkstra's algorithm. In *7th Workshop on Experimental Algorithms (WEA)*, volume 5038 of *LNCS*, pages 319–333. Springer, 2008.
4. D. Delling. Time-dependent sharc-routing. In *16th Annual European Symposium on Algorithms*, volume 5193 of *LNCS*, pages 332–343. Springer, 2008.
5. D. Delling, P. Sanders, D. Schultes, and D. Wagner. Engineering route planning algorithms. In *Algorithmics of Large and Complex Networks*, volume 5515 of *LNCS State-of-the-Art Survey*, pages 117–139. Springer, 2009.
6. R. Geisberger, M. Kobitzsch, and P. Sanders. Route planning with flexible objective functions. In *11th Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2010. accepted.
7. P. Sanders, D. Schultes, and C. Vetter. Mobile route planning. In *16th European Symposium on Algorithms*, pages 732–743, 2008.
8. D. Schultes and P. Sanders. Dynamic highway-node routing. In *6th Workshop on Experimental Algorithms (WEA)*, volume 4525 of *LNCS*, pages 66–79. Springer, 2007.

LTL Satisfiability Checking for Requirements Engineering

Viktor Schuppan (Fondazione Bruno Kessler - Trento, IT)

Linear Temporal Logic (LTL) is a popular variant for formally describing requirements of hardware and software products. In that context tool support, in particular, decision procedures, for LTL needs to address three issues: performance, expressiveness, and feedback to the user. In this talk I address ongoing research w.r.t. the first and third question. First I discuss preliminary experimental results of comparing solvers for LTL satisfiability based on fairly diverse techniques. In the second part I briefly summarize notions of unsatisfiable cores for LTL, i.e., given an inconsistent specification, how to point out parts of the specification that cause an inconsistency.

The first part of the talk is unpublished at the time of writing this abstract. For more information contact schuppan@fbk.eu. For the second part see [Sch09].

References

Sch09. Viktor Schuppan. Towards a notion of unsatisfiable cores for LTL. In Farhad Arbab and Marjan Sirjani, editors, *FSEN*, volume 5961 of *Lecture Notes in Computer Science*, pages 129–145. Springer, 2009.

Keywords: LTL satisfiability, unsatisfiable cores for LTL

Joint work of: Schuppan, Viktor; Darmawan, Luthfi; Roveri, Marco

Comparison of algorithms for checking emptiness on Büchi automata

Stefan Schwoon (TU München, DE)

We re-investigate the problem of LTL model-checking for finite-state systems, which reduces to checking the emptiness of a Büchi automaton.

Because of its importance, the problem has been intensively investigated, and many different solutions have been developed.

In this work, we concentrate on solutions of the type used in Spin [3], which work on-the-fly, sequentially, exact (rather than approximative), and explicit (rather than symbolic techniques using BDDs). A variety of linear-time algorithms for this exist; nonetheless, subtle design decisions can make a great difference to their performance in practice.

We investigate known solutions and propose some improvements to them.

Algorithms are divided into two categories; nested-DFS algorithms and SCC-based algorithms. The former use less memory than the latter; however, in our setting this advantage becomes insignificant, and SCC-based algorithms, which can find counterexamples more quickly, provide better solutions. Among these, a modification of Couvreur's algorithm [1], which can handle generalized Büchi automata, dominates the competition. We also provide an improved nested-DFS algorithm for cases in which memory usage does remain important (e.g., for bitstate hashing).

We compare the algorithms experimentally on a large, representative benchmark suite, measuring their actual run-time performance. Differences between the algorithms are arbitrarily large on individual counterexamples. When averaging over the entire benchmark suite, the differences are less pronounced due to the dominance of "easy" examples with weak Büchi automata. Still, the best algorithm is faster by about 33 per cent on average. We therefore recommend that, for on-the-fly explicit-state model checking, nested DFS should be replaced by better solutions.

The results were published in [2] and improve on former work in [4].

References

1. Jean-Michel Couvreur.
   On-the-fly verification of linear temporal logic.
   In *Proc. Formal Methods*, LNCS 1708, pages 253–271, 1999.
2. Andreas Gaiser and Stefan Schwoon.
   Comparison of algorithms for checking emptiness on Büchi automata.
   In *Proc. MEMICS*, pages 69–77, 2009.
3. Gerard J. Holzmann. *The Spin Model Checker: Primer and Reference Manual.*
   Addison-Wesley, 2003.
4. Stefan Schwoon and Javier Esparza.
   A note on on-the-fly verification algorithms.
   In *Proc. TACAS*, LNCS 3440, pages 174–190, 2005.

Keywords: LTL model checking, Büchi automata

Joint work of: Gaiser, Andreas; Schwoon, Stefan

Full Paper:
http://www7.in.tum.de/um/bibdb/info/schwoon.GS09a.shtml

Semi-External LTL Model Checking

Pavel Simecek (Masaryk University, CZ)

In this presentation we establish c-bit semi-external graph algorithms, – i.e., algorithms which need only a constant number c of bits per vertex in the internal memory. In this setting, we obtain new trade-offs between time and space for I/O efficient LTL model checking: In comparison to former external memory algorithms [1,2,3], the new algorithm presented in [4] is faster, but its internal memory complexity is proportional to the size of a state space.

First, we design a c-bit semi-external algorithm for depth-first search. To achieve a low internal memory consumption, we construct a RAM-efficient perfect hash function from the vertex set stored on disk.

Perfect hash function construction is based on the algorithm invented by Botelho and Ziviani [5]. With perfect hash function, the search itself stores only one bit per vertex denoting which vertices have already been visited.

Since storage of the perfect hash function takes also only the constant number of bits per state, the overall number of bits per state is constant.

We give a similar algorithm for double depth-first search, which checks for presence of accepting cycles and thus solves the LTL model checking problem. The I/O complexity of the search itself is proportional to the time for scanning the search space. For on-the-fly model checking we apply iterative-deepening strategy known from bounded model checking.

References

1. Edelkamp Stefan, Jabbar Shahid, *Large-scale directed model checking LTL*, Model Checking Software (SPIN), 2006.
2. Barnat Jiří, Brim Luboš, Šimeček Pavel, *I/O Efficient Accepting Cycle Detection*, Computer-Aided Verification (CAV), 2007
3. Barnat Jiří, Brim Luboš, Šimeček Pavel, Weber Michael, *Revisiting Resistance Speeds Up I/O-Efficient LTL Model Checking*, Tools and Algorithms for the Construction and Analysis of Systems (TACAS), 2008
4. Edelkamp Stefan, Sanders Peter, Šimeček Pavel, *Semi-external LTL Model Checking*, Computer Aided Verification (CAV), 2008
5. Botelho Fabiano C., Ziviani Nivio, *External perfect hashing for very large key sets*, Conference on Information and Knowledge Management (CIKM), 2007

Keywords: LTL, model checking, external memory

Joint work of: Edelkamp, Stefan; Sanders, Peter; Simecek, Pavel

Full Paper:
http://www.springerlink.com/content/v822184p1l51w854/

See also: S. Edelkamp, P.Sanders, P. Simecek: Semi-external LTL Model Checking in CAV '08: Proc. of the 20th International Conference on Computer Aided Verification, LNCS, 2008

Scalable Distributed-Memory External Sorting

Johannes Singler (KIT - Karlsruhe Institute of Technology, DE)

There are currently two main ways to handle huge inputs in a cost-efficient manner: keeping most data externally on low cost hard disks, and clustering many inexpensive machines. And the most important nontrivial operation needed for processing such huge data sets is sorting, useful for many applications.

Although there is a lot of previous work on distributed-memory external sorting, the problem is not solved yet. In particular, algorithms used in practice can have very bad behavior for worst-case inputs, whereas all previous theoretical results lead to algorithms that need more than two passes even for easy inputs. Here, we present two new algorithms.

DispersedMergeSort is a conceptually simple variant of multiway mergesort that needs two passes even for inputs whose size is close to the theoretical limit for being sorted with two passes. Its first phase is *run formation*, where *runs* of size of the cumulative memory of the parallel machine are loaded, sorted in parallel, and written back to disk. Next follow one or more global merging phases, in which we achieve efficient operation within the memory bounds by using sophisticated prefetching and buffering techniques. However, this algorithm has relatively large communication overhead and outputs the data in a globally striped fashion, i. e., subsequent blocks of output are allocated on subsequent PEs (processing elements).

CanonicalMergeSort in contrast, needs very little communication and outputs the data in a format more conventional: the first PE gets the smallest elements, the second PE the little larger ones, and so on. The first phase is as in DispersedMergeSort, but phase 2 does the necessary (usually only little) distribution, and phase 3 does fully local merging. At least on the average, and up to small "clean up" costs, this algorithm needs only two passes and communicates elements only once.

We have implemented CanonicalMergeSort in C++ using MPI, the STXXL [1], and the parallel mode of the STL implementation of GCC, which is based on the MCSTL [2]. It works in-place and supports hierarchical parallelism and overlapping of computation and communication with I/O.

It sorts about 564 GB/min on a cluster with 195 8-core nodes and 780 disks, connected by InfiniBand. This result won the "Indy GraySort" category of the renowned SortBenchmark in 2009. Yahoo's competing result of 578 GB/min is only 2.5% faster, but its efficiency is much worse, since it used 17 times the number of nodes.

The full paper can be found at [3].

References

1. Roman Dementiev, Lutz Kettner, and Peter Sanders. *STXXL: Standard Template Library for XXL data sets.*, Software Practice & Experience, 38(6):589-637, 2008.
2. Johannes Singler, Peter Sanders, and Felix Putze. *MCSTL: The Multi-Core Standard Template Library.* In Euro-Par, pages 682-694, 2007.
3. Mirko Rahn, Peter Sanders, and Johannes Singler. *Scalable Distributed-Memory External Sorting.* CoRR, abs/0910.2582, 2009.

Joint work of: Rahn, Mirko; Sanders, Peter; Singler, Johannes

Full Paper:
http://arxiv.org/pdf/0910.2582

Problem-Sensitive Restart Heuristics for the DPLL Procedure

Carsten Sinz (KIT - Karlsruhe Institute of Technology, DE)

Search restarts have shown great potential in speeding up SAT solvers based on the DPLL procedure. However, most restart policies presented so far do not take the problem structure into account. In this paper we present several new problem-sensitive restart heuristics. They all observe different search parameters like conflict level or backtrack level over time and, based on their development, decide whether to perform a restart or not.

Keywords: SAT-Solving, search restarts, real-world problems

Full Paper:
http://www.springerlink.com/content/k9450j7717516373

The Pathfinding System of Dragon Age

Nathan Sturtevant (University of Alberta, CA)

BioWare shipped Dragon Age: Origins in November of 2009 to critical acclaim. In this talk we describe the pathfinding system used by the game, which was developed at the University of Alberta [4].

BioWare had two primary criteria when looking for an improved design for their pathfinding engine. The first was speed: only a few milliseconds are available for pathfinding each frame. The second was memory: nearly all the memory available for pathfinding was already being used, so the memory footprint had to be very small.

These criteria were fulfilled in several ways. Drawing on work on abstraction for pathfinding [2,1,3] we developed what eventually became a two-level abstraction on top of the low-level grid already present in the game. The abstraction is a sparse graph which represents the connectivity between regions in the underlying map. Complete planning is performed at the highest level of abstraction. At all levels below this, planning and smoothing are interleaved.

The memory overhead of this approach is around 100k on the largest maps. An average planning step takes approximately 0.1ms in the engine, which means that many units can plan in a single time step. Thus, this work represents the successful conversion of research into a commercial product that will potentially be enjoyed by millions of people.

References

1. A. Botea, M. Müller, and J. Schaeffer. Near Optimal Hierarchical Path-Finding. *Journal of Game Development*, 1(1):7–28, 2004.
2. Robert C. Holte, T. Mkadmi, Robert M. Zimmer, and Alan J. MacDonald. Speeding up problem solving by abstraction: A graph oriented approach. *Artif. Intell.*, 85(1-2):321–361, 1996.
3. N. Sturtevant and M. Buro. Partial pathfinding using map abstraction and refinement. In *Proceedings AAAI*, 2005.
4. Nathan R. Sturtevant. Memory-efficient abstractions for pathfinding. In *AIIDE*, pages 31–36, 2007.

Keywords: Pathfinding, games

Model Checking on General Purpose Graphics Processors

Damian Sulewski (Universität Bremen, DE)

Given that during the last decade graphics processing units (GPUs) have become very powerful, we accelerate model checking by executing complex operations on the graphics card.

A considerable part of the challenges that arise in algorithms for GPUs is due to the specific architectural differences between GPUs and CPUs. Therefore, before describing our approaches in more detail, we give an overview of the GPU architecture and the Compute Unified Device Architecture (CUDA) [1] programming model by the manufacturer NVIDIA.

We present an approach for parallel probabilistic model checking on GPU described in [2]. For this purpose we exploit the fact that some of the basic algorithms for probabilistic model checking rely on matrix vector multiplication. Since this kind of linear algebraic operations are implemented very efficiently on GPUs, the new parallel algorithm can achieve considerable runtime improvements compared to their counterparts on standard architectures.

The second approach focuses on breadth-first search (BFS) to generate the entire state space for explicit-state model checking using external memory, see [3] for details. Sorting-based external BFS bares three computational intensive tasks, all portable to the GPU. Though, our algorithm divides into three stages executed on the GPU. First, it tests if the transitions for a set of states are enabled, followed by generating all their successors in a parallel scan. Finally, it eliminates duplicates delayed by compressing and sorting large state vector sets.

References

1. Lindholm, Erik and Nickolls, John and Oberman, Stuart and Montrym, John, *NVIDIA Tesla: A Unified Graphics and Computing Architecture*, IEEE Micro, 2008
2. Bošnački, Dragan and Edelkamp, Stefan and Sulewski, Damian, *Efficient Probabilistic Model Checking on General Purpose Graphics Processors*,In Proc. of the 16th International SPIN Workshop on Model Checking Software, 2009
3. Edelkamp, Stefan and Sulewski, Damian, *Edelkamp, Stefan and Sulewski, Damian*,Technical Report Technische Universität Dortmund , 2008

Joint work of: Sulewski, Damian; Edelkamp, Stefan; Bošnački, Dragan

Exploiting Transition Locality in Automatic Verification of Concurrent and Hybrid Systems

Enrico Tronci (Sapienza University of Rome, Italy)

We show experimentally that protocols exhibit *transition locality* [1]. That is, with respect to levels of a Breadth–First state space exploration, state transitions tend to be between states belonging to close levels of the transition graph. We support our claim by measuring transition locality for the set of protocols included in the Murφ [2] verifier distribution.

We present a disk–based verification algorithm that exploits transition locality to decrease disk read accesses thus reducing the time overhead due to disk usage [3]. We have implemented our algorithm within the Murφ verifier and call CMurφ the resulting verifier. Our experimental results show that our disk–based verification algorithm is typically more than 10 times faster than previously proposed disk–based verification algorithms and, even when using 10% of the memory needed to complete verification, CMurφ is only between 1.5 and 5.3 times (3 times on average) slower than (RAM) Murφ with enough memory to complete the verification task at hand.

Finally, we show how CMurφ can be extended with *finite precision real numbers* in order to support formal verification of nonlinear *Discrete Time Hybrid Systems* (DTHS) [4]. We show effectiveness of the proposed approach by presenting experimental results on formal verification of industrial size nonlinear deterministic as well as stochastic DTHSs [5].

References

1. Enrico Tronci, Giuseppe Della Penna, Benedetto Intrigila, and Marisa Venturini Zilli. Exploiting transition locality in automatic verification. In *CHARME*, LNCS 2144, pages 259–274. Springer, 2001.
2. David L. Dill, Andreas J. Drexler, Alan J. Hu, and C. Han Yang. Protocol verification as a hardware design aid. In *Proc. of IEEE Int. Conf. on Computer Design on VLSI in Computer & Processors*, pages 522–525. IEEE Computer Society, 1992.
3. G. Della Penna, B. Intrigila, I. Melatti, E. Tronci, and M. Venturini Zilli. Exploiting transition locality in automatic verification of finite state concurrent systems. *STTT*, 6(4):320–341, 2004.
4. Giuseppe Della Penna, Benedetto Intrigila, Igor Melatti, Michele Minichino, Ester Ciancamerla, Andrea Parisse, Enrico Tronci, and Marisa Venturini Zilli. Automatic verification of a turbogas control system with the murphi verifier. In *HSCC*, LNCS 2623, pages 141–155. Springer, 2003.
5. G. Della Penna, B. Intrigila, I. Melatti, E. Tronci, and M. Venturini Zilli. Finite horizon analysis of markov chains with the murφ verifier. *STTT*, 8(4):397–410, 2006.

Keywords: Model Checking, Protocol Verification, Hybrid Systems Verification

Joint work of: Tronci, Enrico; Della Penna, Giuseppe; Intrigila, Benedetto; Melatti, Igor; Zilli, Marisa

The Causal Graph Revisited for Directed Model Checking

Martin Wehrle (Universität Freiburg, DE)

Directed model checking is a well-established technique to tackle the state explosion problem when the aim is to find error states in large systems. In this approach, the state space traversal is guided through a function that estimates the distance to nearest error states. States with lower estimates are preferably expanded during the search. Overall, directed model checking has proved to be a successful approach. However, its success crucially depends on the applied distance function to guide the search. The challenge is to develop distance functions that are efficiently computable on the one hand and as informative as possible on the other hand.

In this work, we introduce the *causal graph* structure to the context of directed model checking [1]. We model systems in terms of parallel processes with global synchronization. The processes are given as directed labeled graphs, consisting of local states and local transitions. In this model, the causal graph is a dependency structure that represents how the processes depend on each other. A process p depends on a process p' if there might be a need to change a local state in p' in order to change a local state in p such that p and p' can synchronize on a common synchronization label.

Based on causal graph analysis, we first propose a distance estimation function for directed model checking. This distance function is an adaptation of the causal graph heuristic that has first been proposed by Helmert in the context of AI planning [3,2]. The causal graph heuristic estimates distances to global error states in the parallel system by computing cost estimates for each process p from the current local state to a nearest local error state. In comparison to earlier approaches, we take into account the synchronization behaviour of p with processes on which p depends (i.e., p's predecessors in the causal graph). This yields a more accurate distance estimate than, e.g., the plain graph distance, as the change of *one* local state in p may depend on complex synchronization behaviour with p's causal predecessors.

Furthermore, we investigate an abstraction technique called *safe abstraction* that is guaranteed to preserve error states. The basic idea is to identify processes of the system that do not depend on any other process on the one hand, and do not have dead ends on the other hand. Such processes are *safe* in the following sense. Abstract error traces π in systems where safe processes are abstracted away can be extended to concrete error traces in polynomial time, as spurious transitions in π can only be caused by safe processes. According to the definition of a safe process, the local states needed to resolve the spurious transition are always reachable.

We have implemented our approaches into the MCTA model checker [4]. The experimental evaluation shows the practical potential of these techniques on large and complex industrial case studies. The causal graph heuristic shows to be competitive with previously proposed distance functions in the context of directed model checking. Moreover, the model reduction obtained by safe abstraction leads to strong performance gains when applicable, and needs only little computation time.

References

1. Wehrle, Martin, Helmert, Malte, *The Causal Graph Revisited for Directed Model Checking*, Proceedings of the 16th International Symposium on Static Analysis (SAS 2009), 2009
2. Helmert, Malte, *The Fast Downward Planning System*, Journal of Artificial Intelligence Research, 2006
3. Helmert, Malte, *A Planning Heuristic Based on Causal Graph Analysis*, Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS 2004), 2004
4. Kupferschmid, Sebastian, Wehrle, Martin, Nebel, Bernhard, Podelski, Andreas, *Faster than* UPPAAL*?*, Proceedings of the 20th International Conference on Computer Aided Verification (CAV 2008) , 2008

Joint work of: Wehrle, Martin; Helmert, Malte

Hierarchical Adaptive State Space Caching based on Level Sampling

Anton Wijs (INRIA Rhne-Alpes, FR)

In the past, several attempts have been made to deal with the state space explosion problem by equipping a search algorithm with a state cache, or by avoiding collision detection, thereby keeping the state hash table at a fixed size [1,2,3,5,6,7,8,9,11,12,13,14].

Most of these attempts are tailored specifically for depth-first search (DFS), and are often not guaranteed to terminate and/or to exhaustively visit all the states.

In this paper [10], we propose a general framework of hierarchical caches which can also be used by breadth-first searches (BFS).

Our method, based on an adequate sampling of BFS levels during the traversal, guarantees that the BFS terminates and traverses all transitions of the state space.

We define several (static or adaptive) configurations of hierarchical caches and we study experimentally their effectiveness on benchmark examples of state spaces and on several communication protocols, using a generic implementation of the cache framework that we developed within the CADP toolbox [4].

References

1. Behrmann, Gerd, Larsen, Kim, Pélanek, Radek, *To Store or Not To Store*, CAV 2003, 2003
2. Courcoubetis, Constantin, Vardi, Moshe, Wolper, Pierre, Yannakakis, Mihalis, *Memory-Efficient Algorithms for the Verification of Temporal Properties*, Formal Methods in System Design, 1992
3. Della Penna, Giuseppe, Intrigila, Benedetto, Tronci, Enrico, Venturini Zilli, Marisa, *Exploiting Transition Locality in the Disk Based Murphi Verifier*, FMCAD 2002, 2002
4. Garavel, Hubert, Lang, Frédérique, Mateescu, Radu, Serwe, Wendelin, *CADP 2006: A Toolbox for the Construction and Analysis of Distributed Processes* CAV 2007, 2007
5. Geldenhuys, Jaco, *State Caching Reconsidered* SPIN 2004, 2004
6. Godefroid, Patrice, Holzmann, Gerard, Pirottin, Didier, *State-Space Caching Revisited*, Formal Methods in System Design, 1995
7. Holzmann, Gerard, *Automated Protocol Validation in Argos, assertion proving and scatter searching*, IEEE Transactions on Software Engineering, 1987
8. Holzmann, Gerard, *An Improved Protocol Reachability Analysis Technique*, Software - Practice and Experience, 1988
9. Jard, Claude, Jéron, Thierry, *Bounded-memory Algorithms for Verification On-the-fly*, CAV 1991, 1991
10. Mateescu, Radu, Wijs, Anton, *Hierarchical Adaptive State Space Caching Based on Level Sampling*, TACAS 2009, 2009

11. Stern, Ulrich, Dill, David, *Combining State Space Caching and Hash Compaction*, 4. GI/ITG/GME Workshop, 1996
12. Stern, Ulrich, Dill, David, *A New Scheme for Memory-Efficient Probabilistic Verification* FORTE 1996, 1996
13. Tronci, Enrico, Della Penna, Giuseppe, Intrigila, Benedetto, Venturini Zilli, Marisa, *Exploiting Transition Locality in Automatic Verification*, CHARME 2001, 2001
14. Tronci, Enrico, Della Penna, Giuseppe, Intrigila, Benedetto, Venturini Zilli, Marisa, *A Probabilistic Approach to Automatic Verification of Concurrent Systems*, APSEC 2001, 2001

Keywords: Explicit state space generation, state space caching, BFS, DFS

Joint work of: Mateescu, Radu; Wijs, Anton

Full Paper:
http://hal.archives-ouvertes.fr/docs/00/38/16/82/PDF/Mateescu-Wijs-09.pdf

See also: R. Mateescu, A.J. Wijs. Hierarchical Adaptive State Space Caching based on Level Sampling, in: Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS '09 (York, UK), S. Kowalewski, A. Philippou (editors), Lecture Notes in Computer Science, vol. 5505, pp. 215-229. Springer, Heidelberg (2009)

Maintaining External Memory Efficient Hash Tables

Philipp Woelfel (University of Calgary)

We present randomized algorithms which maintain hash tables efficiently under circumstances typical for applications dealing with massive data. Consider a set S of n keys from a finite universe U. A hash function $h :\rightarrow \{0, \ldots, m-1\}$ is called *perfect for* S if it is injective on S. If in addition $m = |S|$, then h is called *minimal perfect*.

Fredman, Komlós, and Szemerédi [2] were the first to devise an algorithm which constructs a perfect hash function (with $O(n \log n)$ bits) in expected linear time, and since then minimal perfect hashing has been studied extensively. Pagh [3] showed how to construct in linear time a minimal perfect hash function which can be evaluated very efficiently with simple arithmetics (essentially one or two multiplications) and by probing only one word in external memory. The hash function itself can be encoded in $(2+\epsilon) \cdot n \cdot \log n$ bits. Dietzfelbinger and Hagerup [1] improved Pagh's scheme so that the resulting hash function can be encoded with $(1+\epsilon) \cdot n \cdot \log n$ bits.

We present a dynamic variant of Pagh's scheme. Using exactly the same hash functions, we show how to perform updates in expected amortized constant time. In addition to the $(2+\epsilon) \cdot n \cdot \log n$ bits for encoding of the hash function, for updates we need an auxiliary data structure which comprises $(3+\epsilon) \cdot n \cdot \log n$

bits. We also show that it is possible to reduce the encoding size of the hash functions and the space for the auxiliary data structure to $O(n \log \log n)$ bits. In the dynamic case we obtain a utilization of $1 - \epsilon$, for arbitrary small $\epsilon > 0$. In the static case we still achieve 100% utilization. For both implicit versions the corresponding hash functions can be evaluated in constant time and by probing $O(1)$ consecutive words from external memory.

References

1. M. Dietzfelbinger and T. Hagerup. Simple minimal perfect hashing in less space. In *Proc. of 9th ESA*, volume 2161 of *LNCS*, pp. 109–120. 2001.
2. M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *J. of the ACM*, 31:538–544, 1984.
3. R. Pagh. Hash and displace: Efficient evaluation of minimal perfect hash functions. In *Proc. of 6th WADS*, volume 1663 of *LNCS*, pp. 49–54, 1999.

Dynamic State-Space Partitioning in External-Memory Graph Search

Rong Zhou (PARC - Palo Alto, US)

State-of-the-art external-memory graph search algorithms rely on a hash function, or equivalently, a state-space projection function, that partitions the stored nodes of the state-space search graph into groups of nodes that are stored as separate files on disk. The scalability and efficiency of the search depends on properties of the partition: whether the number of unique nodes in a file always fits in RAM, the number of files into which the nodes of the state-space graph are partitioned, and how well the partitioning of the state space captures local structure in the graph. All previous work relies on a static partitioning of the state space. In this paper, we introduce a method for dynamic partitioning of the state-space search graph and show that it leads to substantial improvement of search performance.

Keywords: External-memory graph search, heuristic search

Joint work of: Zhou, Rong; Hansen, Eric A.

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2010/2433

Acknowledgement

Thanks to Damian Sulewski for chasing the authors to add references to their abstract and for typsetting this paper. Thanks to Jutta and Boris Goldammer for their support to initiate discussions among the participants.