Data Stream Query Optimization Across System Boundaries of Server and Sensor Network

Wolfgang Lindner
MIT, CSAIL
32 Vassar ST, Cambridge, MA 02139, USA
wolfgang@csail.mit.edu

Holger Velke, Klaus Meyer-Wegener
FAU Erlangen-Nuernberg, Informatik 6
Martensstr. 3, 91058 Erlangen, Germany
sihovelk@stud.uni-erlangen.de, kmw@acm.org

Abstract

One of the most important input providers for data stream management systems (DSMSs) is a sensor network. Such a network can have query functionality offered as a sensor network query processor (SNQP). Then some of the data stream operators can be executed in the DSMS as well as in the SNQP. This paper addresses the problem of finding the optimal solution. It shows first steps like the moving of operators and the modification of the epoch duration. The primary goal is to prolong the lifetime of the sensor network. A QoS-based goal function is introduced, and the optimization process is explained. It has been implemented with Borealis and TinyDB. Some preliminary results from the ongoing evaluation are given.

1. Introduction

Over the past years, sensor network query processors (SNQPs) like Cougar [10] and TinyDB [7] have been developed to ease the access to data produced by sensor networks (SNs). Such networks may consist of small battery-powered motes with possibly lossy wireless communication. The main goal for these SNs is to reduce the power consumption in order to increase the lifetime of the motes. This paper focusses on such SNs of battery-powered motes.

SNQPs act as a natural data source for data stream management systems (DSMSs). Today, the major DSMSs are STREAM [4], TelegraphCQ [5], and Borealis [2]. Adabi et al. developed an integration framework [3] for DSMSs and SNQPs. Integrating these systems is promising because it extends the query execution from the DSMS to the SNQP, so more flexibility is added to the query optimization. To date the integration framework features only a very basic optimization for allocating operators to the two systems. This paper presents a more elaborated approach to that and uses Borealis and TinyDB as platforms.

The problem of allocating operators in the SNQP rather than in the DSMS can be faced with two approaches. First, the basic approach is to move operators between the two systems without changing their order in the query. Second, an extended approach is to additionally reorder the operators of the query. Obviously, with reordering of operators there are more possibilities of allocating the operators in the combined system, yielding optimal results. However, the intention of this paper is to show that even with the basic and much simpler approach of moving operators without changing their order in the query, the lifetime of the SN can be increased significantly. In a next step reordering can be included. Operator reordering in the context of SNQPs is already considered by Srivastava et al. [9].

The purpose of extending the query processing to the SNQP is twofold. First, Borealis is enabled to better fulfill the QoS requirements of the users. Second, the lifetime of the SN is increased by executing tuple-reducing operators in the network.

The ideas and concepts which are the basis for the optimization are clarified in the next section. In Section 3 the new optimization possibilities for DSMSs with an integrated SNQP are outlined. The QoS-based rating model is pointed out in Section 4. In Section 5 the optimization procedure is presented. Section 6 presents an evaluation of the prototypical implementation. The paper ends with a conclusion.

2. Background

2.1. Query-processing Operators: the Borealis Boxes

The Borealis system [11] uses the box-and-arrow paradigm to specify the data flow. A *box* represents a query operator and an arrow represents the data flow between boxes. In this paper box is used as a synonym for operator.

Borealis has inherited its set of boxes from Aurora [1].

From this set, the following are of interest for the optimization, because they are available in most SNQPs, too:

- `filter` is equivalent to an extended relational selection. It can split an input stream into one or more output streams depending on predicates.
- `map` is equivalent to a relational projection. It may also perform computations on attributes of the input tuples.
- `aggregate` applies aggregate functions to a sliding window of stream tuples.

For the purpose of this paper an additional user-defined box named `static join` is used. It joins the tuples of its input stream with those of a static table.

2.2. Sensor Network Query Processors

SNQPs provide database-like access to sensor data in SNs via a simple querying interface. Today there are two important SNQPs: Cougar [10] developed at Cornell University and TinyDB [7] from UC Berkeley.

Both systems have important properties in common. They use SNs of battery-powered motes connected by wireless communication. Communication with the outside world (e.g. a DSMS) is directed through a base station. Depending on the surrounding conditions, the communication can be lossy, i.e. tuples may get lost when sent from the motes to the base station. Also, the motes have only a limited amount of energy. Energy is consumed by communicating, processing and sensing. However, the energy consumption of communication is dominant [8]. Due to this fact processing on the motes that reduces communication is worthwhile.

In both systems queries are specified using an SQL-like query language such as TinySQL [7]. A query is deployed at the base station and from there propagated into the network of sensing motes. Either system supports the following set of relational operators: selection, projection, aggregation, and static join.

SNs acquire data in certain intervals. The sensing rate for a query is specified as the time span between subsequent sensings. In TinySQL, this sampling interval is called *epoch duration*.

Compared with Borealis, the functionality of SNQPs is limited by the SQL-like query language. Only operators with one input stream and one output stream are available. Basically, the supported Borealis boxes are `filter` (F), `map` (M), `aggregate` (A), and `static join` (J). But due to the limited functionality of the SNQPs, they only come as restricted versions. For example, a Borealis `filter` box supports several output streams. To execute it in the SNQP, it must be translated into a semantically equivalent set of `selection` queries. Both SNQPs support only `map` boxes with a simple projection. Further restrictions specific to a certain SNQP will not be enumerated here. From now on, only the restricted versions of boxes will be considered, since they are eligible for execution in the SNQP and thus subject to optimization.

2.3. Integration Framework

A framework for the integration of SNQPs with Borealis has been proposed in [3]. It offers an architecture for integrating data sources with query capabilities and a system with metrics and scores for measuring the integrated data sources.

The proposed architecture consists of wrappers and a connection layer. Each *wrapper* covers a specific integrated data source and provides a uniform way to access it. It further offers information about services and constraints of the SNQP. Services are the available data and the supported Borealis boxes. Constraints define boundaries for scores. The wrapper also offers information about the actual system state at runtime. The *connection layer* consists of sensor proxies. A *sensor proxy* connects one Borealis site with one or more data sources covered by wrappers.

Metrics and scores have first been introduced in [6]. *Metrics* are query- or system-dependent measurements taken with a sampling interval that equals the epoch duration. They represent the system behavior during the sampling interval or at the sampling time.

In detail, the wrappers provide the following metrics. The first two metrics (tl and tps) are system-dependent. The remaining metrics are query-dependent.

- TransmissionsLeft (tl): The remaining number of transmissions from the SN.
- Transmissions (tps): The number of transmissions per second from the SN.
- Throughput (tp): The number of tuples per second for a certain query.
- Sent (s): The number of tuples sent by the motes for a certain query.
- Received (r): The number of tuples received by the base station for a certain query. It can be less than s since the communication is lossy.
- Selectivity (se): The selectivity of a certain query in the SN.

tl represents the remaining energy of the motes in terms of the number of transmissions until the SN is out of power. tps gives the number of transmissions per second sent from the SN to the base station including the tuples of all queries in the network. tp represents the transmissions per second for a particular query. se finally indicates the fraction of transmitted tuples from the sensed tuples.

Based on metrics, *scores* can be defined as combined values that indicate the quality of processing. Here, the following scores are used:

- Lifetime (LIF): The remaining time until the SN is out of power.
  $LIF = \frac{tl}{tps}$
- Throughput (THR): The number of sensed tuples per second for a certain query. The value represents the input tuple rate of the SNQP.
  $THR = \frac{tp \cdot s}{se \cdot r}$
- Coverage (COV): The fraction of successfully received tuples for a certain query.
  $COV = \frac{r}{s}$

3. Optimization Possibilities

Borealis is a powerful data stream processing system. An SNQP in contrast has less processing capabilities and most notably, it has only a limited amount of energy. Therefore, the main goal of the optimization should be to save energy at the motes.

3.1. Optimization Tasks

The optimization has to decide which part of a query is executed in the SNQP. For this subquery, the regular interface for querying the SNQP, namely the SQL-like query language must be used. It allows to specify the operators for the query and additionally the epoch duration. The following two aspects are addressed here as optimization tasks: First, the allocation of boxes to the two systems is optimized. Second, the epoch duration for the query in the SNQP is optimized.

The optimization of the box allocation decides which boxes are processed by the SNQP rather than in Borealis. Boxes are moved to the SN only to reduce the number of tuples that have to be transmitted within the SN. In general the number of transmissions is reduced by boxes with a selectivity less than 1. But boxes with such a selectivity should not always be processed in the SN; this depends on the particular box. The box-specific aspects are described below in 3.2.

A Borealis query does not specify an epoch duration. But for a query in an SNQP an epoch duration must be given. It can be computed based on the user requirements (see Section 4) as a trade-off between lifetime and throughput. A high epoch duration results in a high lifetime but a low throughput and vice versa. Additionally, the epoch duration can be used for load shedding: Instead of dropping tuples, the epoch duration is increased.

3.2. Details on Boxes

The boxes can be classified in two categories. The first includes boxes that are generally executed in the SN: `map` and `filter`. The second category contains boxes whose allocation is determined by the optimizer: `aggregate` and `static join`.

A `filter` box has a selectivity less than or equal to 1. If moved to the SN, it reduces the number of transmissions (unless the selectivity is 1). This reduction decreases power consumption and increases the lifetime of the SN. A `filter` executed in an SN with a certain tuple loss rate has no effect on the coverage of the query results. As a consequence, it is always executed there.

The `map` box has a selectivity of 1. Due to the projection of attributes, the size of the tuples can be reduced. Hence, moving a `map` to the SN can reduce the amount of data to transmit. In that case fewer sensors of the motes have to be read, too. This decreases the consumed power. Executing a `map` in an SN with lossy communication will not affect the coverage of the query because the number of tuples to transmit is not changed. So, `map` is always executed there.

The properties of a `static join` depend on the static table and the join predicates. The selectivity can theoretically be between 0 and ∞. It depends on the sensed data and on the data in the table. Based on them it is possible to decide whether the selectivity can be greater than 1. The tuple size can be enlarged. Moving a `static join` to the SN requires initial transmission of the static table to the motes, which consumes power and reduces the lifetime of the SN. If the `static join` has a selectivity less than 1, it reduces the number of transmissions within the SN and thus improves the lifetime. The `static join` must be executed in the SN long enough to amortize the initial cost of table distribution. Executing a `static join` in an SN with lossy communication does not affect the coverage of the query.

An `aggregate` box cumulates tuples. It has a selectivity less than 1. So an `aggregate` executed in the SN reduces the number of transmissions and therefore increases the lifetime. The loss of an aggregate tuple implies the loss of all tuples contributing to the aggregate. Therefore the loss of an aggregate tuple results in a low coverage score, whereas a successfully transmitted aggregate tuple results in a high coverage score. So the coverage has an increased variance if the `aggregate` is executed in an SN with lossy communication. An `aggregate` should be executed in the SN if the network is not lossy or the increased variance of coverage is acceptable for the user.

4. Rating Model

Borealis users define their QoS requirements for the output of queries in terms of lifetime, latency, throughput, and coverage. These requirements must be transformed upstream, so they can be used for the optimization. Latency cannot be supported at the moment.

For evaluation purposes a simplified way of specifying user QoS requirements is used here. For each score (lifetime, throughput, and coverage), two values representing the upper and lower QoS boundary are given. Scores below the lower boundary are not acceptable, whereas scores above the upper boundary indicate best QoS for the user. With these two values a *QoS function* is defined. The domain for it is set to $[0, 1]$, where 1 indicates best QoS and 0 indicates worst. A QoS function transforms a score into a QoS value. For example, the QoS function for coverage QoS_{COV} with the lower boundary COV_{low} and the upper boundary COV_{up} can be:

$$QoS_{COV} = \begin{cases} 0 & \text{if } COV < COV_{low} \\ 1 & \text{if } COV > COV_{up} \\ \frac{COV - COV_{low}}{COV_{up} - COV_{low}} & \text{else} \end{cases}$$

The QoS functions for the other scores are defined in a similar fashion. They are used to rate the box allocations and epoch durations for the optimization. Those with the highest QoS are selected as the best solution.

5. Optimization Process

When a new query is deployed, the distribution is initially optimized. Since no metrics are available yet, only maps and filters are moved to the SN. The initial epoch duration is set to $ed = n_{motes}/tp_{up}$ using the number of motes n_{motes} in the SN and the throughput requirement by the user, namely the upper throughput boundary tp_{up}. Once running, each query in the SN is monitored individually. The monitor checks whether the user QoS requirements are met and no system constraints are violated. In addition the monitor recognizes optimization opportunities. This is for example a `static join` with a high selectivity running in Borealis. When user or system requirements are violated or an optimization opportunity is recognized, the monitor triggers dynamic optimization.

The dynamic optimization consists of two independent tasks: box allocation and adjustment of epoch duration. Optimizing the box allocation raises the selectivity and finds a balance of coverage and lifetime (see `aggregate`). The epoch duration is optimized to get the best trade-off between lifetime and throughput.

Optimizing the box allocation means to decide which box should be best executed where. As stated above only the allocation of `aggregate` and `static join` must be considered.

The first allocation is the one with none of the considered operators (`aggregate` and `static join`) allocated in the SNQP. In each step, one additional `aggregate` or `static join` is moved to the SNQP. Also, all `maps` and `filters` behind it are moved.

To rate a certain box allocation, metrics must be estimated. The estimation is based on the current metrics plus the current and the new box allocation. It uses the selectivity of the boxes, the size of static join tables and the type of the boxes that are to be moved to the SNQP.

For calculation of selectivity, the selectivity of the whole query boxes currently running in the SNQP is multiplied with the selectivity of the box considered for movement. Borealis has statistics on the selectivity of each box.

When a selective box is moved to the SNQP, this also changes the Transmissions tps, the Throughput tp, the Sent s and the Received r metrics. Therefore these values have to be estimated using their current values tps_{cur}, tp_{cur}, s_{cur}, r_{cur} and the current and estimated selectivity.

$$\begin{aligned} tp_{est} &= tp_{cur} \cdot sel_{est}/sel_{cur} \\ s_{est} &= s_{cur} \cdot sel_{est}/sel_{cur} \\ r_{est} &= r_{cur} \cdot sel_{est}/sel_{cur} \\ tps_{est} &= tps_{cur} - tp_{cur} + tp_{est} \end{aligned}$$

When a static join is moved to the SN, its table has to be transmitted to the motes of the SN. This causes a certain amount of transmission, depending on the size of the static table. The number of transmissions n_{table} needed for the distribution of the table reduces the tl metric.

$$tl_{est} = tl_{cur} - n_{table}$$

The estimated metrics are used as input for the QoS model to evaluate the new allocation.

Optimization of the epoch duration is a trade-off between lifetime and throughput. For rating a new epoch duration ed_{new}, the resulting Transmissions tps, the Throughput tp, the Sent s, and the Received r metric are estimated:

$$\begin{aligned} tp_{est}(ed_{new}) &= tp_{cur} \cdot ed_{cur}/ed_{new} \\ s_{est}(ed_{new}) &= s_{cur} \cdot ed_{cur}/ed_{new} \\ r_{est}(ed_{new}) &= r_{cur} \cdot ed_{cur}/ed_{new} \\ tps_{est}(ed_{new}) &= tps_{cur} - tp_{cur} + tp_{est} \end{aligned}$$

The resulting changes have influence only on the lifetime and throughput score. The coverage score is not changed, because the Sent and Received metrics are changed at the same ratio.

The QoS values of lifetime and throughput for a query are defined as functions of ed_{new}. They use the lifetime score $LIF(ed_{new})$ and the throughput score $THR(ed_{new})$:

$$\begin{aligned} LIF(ed_{new}) &= \frac{tl_{cur}}{tps_{est}(ed_{new})} \\ &= \frac{tl_{cur}}{tps_{cur} - tp_{cur}(1 - ed_{cur}/ed_{new})} \\ THR(ed_{new}) &= \frac{tp_{est}(ed_{new}) \cdot s_{est}(ed_{new})}{sel_{cur} \cdot r_{est}(ed_{new})} \\ &= \frac{tp_{cur} \cdot s_{cur} \cdot ed_{cur}}{sel_{cur} \cdot r_{cur} \cdot ed_{new}} \end{aligned}$$

The QoS functions for lifetime and throughput need the upper and the lower lifetime boundaries (LIF_{up}, LIF_{low}) and the upper and the lower throughput boundaries (THR_{up}, THR_{low}).

$$QoS_{LIV}(ed_{new}) = \begin{cases} 0 & \text{if } LIF(ed_{new}) < LIV_{low} \\ 1 & \text{if } LIF(ed_{new}) > LIF_{up} \\ \frac{LIF(ed_{new}) - LIF_{low}}{LIF_{up} - LIF_{low}} & \text{else} \end{cases}$$

QoS_{THR} is defined analogously.

With the combined QoS function for lifetime and throughput the overall QoS is calculated. Optimization should maximize the value of this function.

$$QoS(ed_{new}) = \begin{cases} 0 & \text{if } LIF(ed_{new}) < LIF_{low} \\ 0 & \text{if } THR(ed_{new}) < THR_{low} \\ \frac{QoS_{THR}(ed_{new}) + QoS_{LIF}(ed_{new}))}{2} & \text{else} \end{cases}$$

With inverted QoS functions, the epoch durations for the upper or lower user-defined boundaries can be calculated. These are the epoch durations for the lower lifetime boundary ed_{ll}, for the upper lifetime boundary ed_{lu}, for the lower throughput boundary ed_{tl}, and for the upper throughput boundary ed_{tu}.

$$\begin{aligned} ed_{ll} &: \quad LIF(ed_{ll}) = LIF_{low} \\ ed_{lu} &: \quad LIF(ed_{lu}) = LIF_{up} \\ ed_{tl} &: \quad THR(ed_{tl}) = THR_{low} \\ ed_{tu} &: \quad THR(ed_{tu}) = THR_{up} \\ \text{with} & \quad ed_{ll} < ed_{lu} \text{ and } ed_{tu} < ed_{tl} \end{aligned}$$

The epoch duration must be within the interval $[ed_{ll}, ed_{tl}]$. If the epoch duration is less than ed_{ll}, the user requirement for lifetime is violated. If the epoch duration is greater than ed_{tl}, the user requirement for throughput is violated.

Figure 1 illustrates how the QoS values are associated with the epoch duration. Each row (a – f) shows a certain arrangement of the epoch durations that represents the boundaries of the QoS functions. For each interval a certain continuous QoS function is specified. With the derivation of the QoS functions their gradients and extrema are calculated. The dashed line symbolizes gradient and extrema for a certain interval. For example, the illustration in line (a) shows:

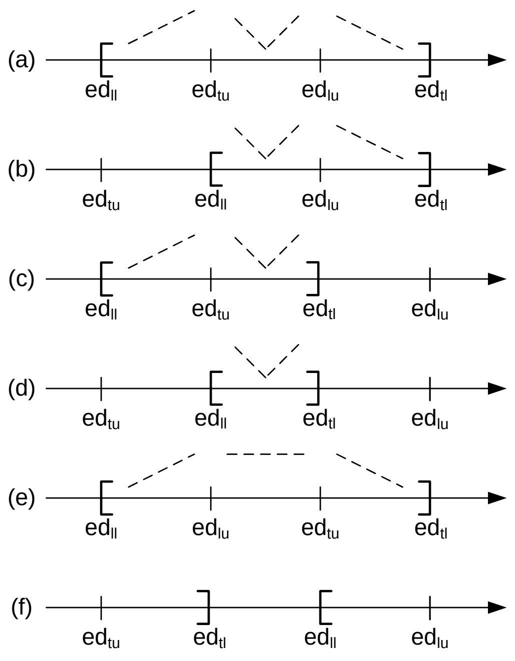

Figure 1. Epoch Duration – QoS Function

- $[ed_{ll}, ed_{tu}]$ – The QoS function is strictly monotonically increasing.
- $[ed_{tu}, ed_{lu}]$ – The QoS function has one minimum. Therefore it is strictly monotonically decreasing before the minimum and strictly monotonically increasing after the minimum.
- $[ed_{lu}, ed_{tl}]$ – The rating function is strictly monotonically decreasing.

The symbols of the other rows (b – f) are to be interpreted in the same way. In line (a), ed_{tu} and ed_{lu} are the candidate epoch durations with a maximum QoS value. So for them the QoS values are calculated and the one with the maximum is selected.

For the cases of the first four lines (a – d), the two candidates for the optimal epoch duration are commonly the maximum of $\{ed_{ll}, ed_{tu}\}$ and the minimum of $\{ed_{lu}, ed_{tl}\}$. The lines (e) and (f) are special cases. Line (e) has the interval $[ed_{lu}, ed_{tu}]$ with a QoS value of 1. Hence, the optimal epoch duration is ed_{tu}. Line (f) shows the case where no epoch duration fulfills both user requirements at a time. So the query must be suspended.

6. Evaluation

The proposed optimization has been implemented with TinyDB. It allows for dynamic optimization of box allocation and epoch duration. To investigate their respective

measurement	box allocation	epoch duration
One	off	off
Two	off	on
Three	on	off
Four	on	on

Table 1. The Measurements

influence, either can be switched on and off. A monitor for queries in TinyDB and boxes in Borealis has also been implemented. TinyDB can only be monitored per query and not per operator. In Borealis, only movable boxes are monitored.

Two queries are used here, one with a `static join`, the other with an `aggregate` box. Since the `map` and `filter` boxes are always allocated to the SN, the optimizer has to decide on the allocation of those boxes only.

For each measurement, the scores and the metrics of the SNQP are recorded per epoch. Here the most interesting is the lifetime score. The four different measurement cases are shown in Table 1. They differ in the optimization tasks being switched on or off.

For space reasons, the diagrams of the measurement results are not included here. Please contact the authors for the full version of the paper.

The results clearly state that the optimization makes the right decision for reallocating `aggregate` and `static join` operators. Each optimization tasks individually yields an improvement of lifetime already. The optimization of the epoch duration obviously has the side-effect of lowering the throughput of the query. The combination of both optimizations produces the best results for both queries. Hence, the proposed optimization makes the right decisions for the considered operator types.

7. Conclusion

In this paper a solution for optimizing the query execution between Borealis and an integrated SNQP has been presented. The optimization is split into optimizing the operator allocation and the epoch duration in the SNQP. It is sufficient to reduce the optimization of the operator allocation to the decision where to execute `aggregate` and `static join` operators. A solution for directly calculating the optimal epoch duration based on the mathematical properties of a QoS-based optimization model has been given. That solution has been evaluated with a prototype which integrates TinyDB with Borealis. Preliminary results show that the optimization rises the lifetime of the sensor network significantly.

Ongoing work addresses the reordering of boxes in a Borealis query in order to move even more to the SNQP. Additional metrics and scores will be introduced, e.g. the latency of a tuple from sensing to the output. Different QoS functions can be defined and tested. And finally, more evaluations are still to be done.

Acknowledgements

The authors thank the anonymous reviewers for their fair feedback and their valuable suggestions. However, the need to shorten the paper left hardly any room to implement them.

References

[1] D. J. Abadi et al. Aurora: A new model and architecture for data stream management. *VLDB J.*, 12(2), 2003.

[2] D. J. Abadi et al. The design of the Borealis stream processing engine. In *Proc. CIDR*, 2005.

[3] D. J. Abadi, W. Lindner, S. Madden, and J. Schuler. An integration framework for sensor networks and data stream management systems. In *Proc. VLDB*, 2004.

[4] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma, and J. Widom. STREAM: The Stanford stream data manager. *IEEE Data Eng. Bull.*, 26(1), 2003.

[5] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *Proc. CIDR*, 2003.

[6] W. Lindner and J. Schuler. Integrating arbitrary constraints into the query optimization process of data stream management systems. Technical report, MIT, 2006.

[7] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proc. ACM SIGMOD*, 2003.

[8] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Commun. ACM*, 43(5):51–58, 2000.

[9] U. Srivastava, K. Munagala, and J. Widom. Operator placement for in-network stream query processing. In *PODS*, 2005.

[10] Y. Yao and J. Gehrke. Query processing in sensor networks. In *Proc. CIDR*, 2003.

[11] S. B. Zdonik, M. Stonebraker, M. Cherniack, U. Çetintemel, M. Balazinska, and H. Balakrishnan. The Aurora and Medusa projects. *IEEE Data Eng. Bull.*, 26(1):3–10, 2003.