s c c h
software competence center
hagenberg

FUZZY.LOGIC.LAB.LINZ

Advances in Knowledge-Based Technologies

Proceedings of the Master and PhD Seminar Winter term 2013/14, part 1

Softwarepark Hagenberg
SCCH, Room 0/2
19 November 2013

Software Competence Center Hagenberg
Softwarepark 21
A-4232 Hagenberg
Tel. +43 7236 3343 800
Fax +43 7236 3343 888
www.scch.at

Fuzzy Logic Laboratorium Linz
Softwarepark 21
A-4232 Hagenberg
Tel. +43 7236 3343 431
Fax +43 7236 3343 434
www.flll.jku.at

Program

Session 1. Chair: Susanne Saminger-Platz

13:00 Radko Mesiar:
Copula-Based Universal Integrals

13:30 Ciprian Zavoianu:
Enhancing Multi-Objective Evolutionary Algorithms with Decomposition Strategies

14:00 Carlos Cernuda:
Active learning for optimized self-adaptive calibration in viscose production

Session 2. Chair: Bernhard Moser

14:45 Patrick Traxler:
From Fault Detection of Large Amounts of Photovoltaic Systems to Median Regression

15:15 Francisco Serdio:
Fault Detection in Multi-Sensor Networks

15:45 Werner Reisner:
Parallelization of Algorithms for Linear Discrete Optimization using ParaPhrase

Copula-Based Universal Integrals

Radko Mesiar

Slovak University of Technology
Faculty of Civil Engineering
Bratislava, Slovakia
UTIA CAS, Pod vodárenskou věží 4
182 08 Praha, Czech Republic
radko.mesiar@stuba.sk

This work was done in collaboration with E.P. Klement, JKU Linz, F. Spizzichino, University of Roma La Sapienza, and A. Stupňanová, STU Bratislava. As its main result, a hierarchical family of copula-based integrals is introduced and discussed. When considering the product copula, a graded family of decomposition integrals independently introduced by Even and Lehrer, and by Mesiar and Stupňanová, is recovered. Boundary members are distinguished universal integrals introduced by Klement at al. in 2010.

Acknowledgment

The work on this presentation was supported by grants VEGA 1/0143/11 and GAČR P 402/11/0378.

References

1. B. Bassan and F. Spizzichino, *Relations among univariate aging, bivariate aging and dependence for exchangeable lifetimes.* J. Multivariate Anal. 93 (2005) 313–339.
2. G. Beliakov and S. James, *Citation-based journal ranks: the use of fuzzy measures,* Fuzzy Sets and Systems 167 (1) (2011) 101–119.
3. G. Choquet, *Theory of capacities.* Ann. Inst. Fourier 5 (1953/54) 131–295.
4. F. Durante and C.Sempi, *Semicopulæ,* Kybernetika 41 (3) (2005) 315–328.
5. Y. Even and E. Lehrer, *Decomposition-Integral: Unifying Choquet and the Concave Integrals.* Econ. Theory, 2013, DOI 10.1007/s00199-013-0780-0.
6. E.P. Klement, R. Mesiar,E. Pap, *A universal integral as common frame for Choquet and Sugeno integral.* IEEE Transactions on Fuzzy Systems 18 (2010) 178–187.
7. H. Lebesque, *Intégrale, longueur, aire.* Habilitation thesis, Université de Paris, Paris, 1902.
8. R. Mesiar and A. Stupňanová, *Decomposition integrals*, International Journal of Approximate Reasoning 54 (2013) 1252–1259.
9. R. B. Nelsen, *An Introduction to Copulas.* Springer, New York (second edition), 2006.
10. N. Shilkret, *Maxitive measure and integration.* Indag. Math. 33 (1971) 109–116.
11. A. Sklar, *Fonctions de répartition à n dimensions et leurs marges,* Publ. Inst. Statist. Univ. Paris 8 (1959) 229–231.
12. M. Sugeno, *Theory of fuzzy integrals and its applications,* PhD thesis, Tokyo Institute of Technology, 1974.
13. V. Torra and Y. Narukawa, *The h-index and the number of citations: two fuzzy integrals.* IEEE Trans. on Fuzzy Systems 16 (3) (2008) 795–797.

Enhancing Multi-Objective Evolutionary Algorithms with Decomposition Strategies

Alexandru-Ciprian Zăvoianu [a,c] Edwin Lughofer [a]
Wolfgang Amrhein [a,b] Erich Peter Klement [a,c]

[a] *Department of Knowledge-based Mathematical Systems / Fuzzy Logic Laboratory Linz-Hagenberg, Johannes Kepler University of Linz, Austria*

[b] *Institute for Electrical Drives and Power Electronics, Johannes Kepler University of Linz, Austria*

[c] *LCM, Linz Center of Mechatronics, Linz, Austria*

Abstract

Decomposition-based multi-objective evolutionary algorithms like MOEA/D [1] propose a novel search paradigm that is not based on the Pareto-based elitism mechanism pioneered by earlier algorithms (NSGA-II [2] and SPEA2 [3]). The main idea of the former is to split the multi-objective optimization problem in many single-objective optimization problems that are to be solved simultaneously. These decomposition strategies seem to be especially sucessful with problems that have complicated Pareto fronts [4].

In the present work we are describing several attempts to incorporate decomposition strategies into a previously proposed algorithm (i.e., DECMO [5]) in order to further improve stability and performance.

Preliminary results indicate that a carefully chosen decomposition strategy is generally able to improve the overall quality (in terms of hypervolume [6]) of the Pareto fronts produced by DECMO on a test bench of 20 artificial test problems. This increase of the hypervolume values obtained in the last generation comes at a price of a slightly slower convergence speed. Nevertheless, the resulting algorithm remains fairly robust and its performance is quite good even when comparing with state-of-the-art methods like MOEA/D-DE (with the CEC 2009 parameter settings).

Key words: multi-objective optimization, decomposition strategies, evolutionary algorithms, cooperative coevolution, differential evolution.

References

[1] Q. Zhang, H. Li, MOEA/D: A multiobjective evolutionary algorithm based on decomposition, Evolutionary Computation, IEEE Transactions on 11 (6) (2007) 712–731.

[2] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 6 (2) (2002) 182–197.

[3] E. Zitzler, M. Laumanns, L. Thiele, SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization, in: Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN 2001), International Center for Numerical Methods in Engineering (CIMNE), 2002, pp. 95–100.

[4] H. Li, Q. Zhang, Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II, IEEE Transactions on Evolutionary Computation 13 (2) (2009) 284–302.

[5] A.-C. Zăvoianu, E. Lughofer, W. Amrhein, E. P. Klement, Efficient multi-objective optimization using 2-population cooperative coevolution, in: International Conference on Computer Aided Systems Theory (EUROCAST 2013), Springer Berlin / Heidelberg, In press.

[6] M. Fleischer, The measure of Pareto optima. applications to multi-objective metaheuristics, in: International Conference on Evolutionary Multi-Criterion Optimization (EMO 2003), Springer, 2003, pp. 519–533.

Active Learning for Optimized Self-Adaptive Calibration in Viscose Production

Carlos Cernuda[a], Edwin Lughofer[a], Georg Mayr[b], Thomas Röder[c], Peter Hintenaus[d], Wolfgang Märzinger[e]

[a]*Department of Knowledge-Based Mathematical Systems, Johannes Kepler University Linz, Austria*
[b]*Kompetenzzentrum Holz GmbH, Linz, Austria*
[c]*Lenzing AG, Lenzing, Austria*
[d]*Software Research Center, Paris Lodron University Salzburg, Austria*
[e]*i-RED Infrarot Systeme GmbH, Linz, Austria*

Abstract

In viscose production, it is crucial to monitor some process parameters (the concentration of several substances) as part of the spin bath in order to assure a high quality of the final product. During on-line production, these process parameters usually show a quite high dynamics depending on which fibre type is produced and depending on environmental influences. Thus, conventional chemometric models (e.g. Principal Components Regression, Partial Least Squares Regression or Locally Weighted Regression) as well as new non-linear approaches recently employed in calibration (like FLEXFIS, Flexible Fuzzy Inference Systems [1]), which are trained based on collected calibration spectra from Fourier transform near infrared (FT-NIR) measurements and kept fixed during the whole life-time of the on-line process, show a quite imprecise and unreliable behavior when predicting the concentrations of new on-line data. Recently, a new concept named eChemo , evolving chemometric models, was developed which obtained really good results on real world data [2]. It possesses the ability to self-adapt and re-calibrate based on newly recorded on-line data, but unfortunately it requires permanent supervision, i.e. real values measured by means of a titration automaton, which is too time intensive and too expensive from an industrial point of view. We propose here an alternative approach, based on self-adaptive calibration models within a sliding window concept, which brings more flexibility to the incremental learning phase and especially also in the forgetting process of older samples. It consists in a window, containing W samples, that is updated every R incoming new samples by means of substitution of one sample by the new one. The selection of the outgoing sample is guided by a two-stage selection process: i) check if the new sample is significantly similar to any sample in the current window, by means of a spectral similarity measure based on Bayesian statistics [3]. If so, we substitute the most similar

one. ii) If not, we create W models leaving a different sample out, thus we will choose the sample out of the best model as the less informative one. Therefore, that will be the outgoing sample. This approach permits introducing more flexibility in terms of self-adjusting learning parameters and self-optimizing input dimensionality for the self-adaptive calibration models, thus allowing the integration of input structure changes, which was not possible in eChemo. Moreover, it also gets very accurate results and requires much less real values, leading to a huge computational time and financial saving. We show that our approach significantly outperforms conventional S-o-A models, and eChemo, when using real world data streams.

Keywords: Active learning, on-line modelling, cost optimization, multivariate calibration, viscose production

---

[1] E. Lughofer, FLEXFIS: A robust incremental learning approach for evolving TS fuzzy models, IEEE Trans. on Fuzzy Systems 16 (6) (2008) 1393–1410.

[2] C. Cernuda, E. Lughofer, L. Suppan, T. Röder, R. Schmuck, P. Hintenaus, W. Märzinger, J. Kasberger, Evolving chemometric models for predicting dynamic process parameters in viscose production, Analytica Chimica Acta 725 (2012) 22–38.

[3] F. Gan, P. K. Hopke, J. Wang, A spectral similarity measure using bayesian statistics, Analytica Chimica Acta 635 (2) (2009) 157 – 161.

From Fault Detection of Large Amounts of Photovoltaic Systems to Median Regression

Patrick Traxler

Software Competence Center Hagenberg, Austria

We study a model-based approach to detect sustainable faults of photovoltaic systems. We describe models and algorithms which allow us to analyze a large amount of photovoltaic systems. Our approach is based on median regression. We assume that we know the irradiance and produced energy of a photovoltaic system or that we know the energy of photovoltaic systems which are close to each other. The particular challenge of the data analysis problem we study here is the possible huge amount of photovoltaic systems and their continuous streams of data.

We also discuss some abstractions and generalizations of the problem. In particular, we discuss two algorithms for median regression (a.k.a. ℓ_1-regression). These two problems correspond to geometric problems, namely hyperplane fitting. Our first algorithm solves the problem of minimizing least absolute deviations in the plane such that the fitting line goes through the origin. Our second algorithm solves the same problem in three dimensions. In this case, we want a fitting plane which goes through the origin minimizing least absolute deviations. We provide experimental results, in particular, a comparison with simplex-based algorithms, showing that our algorithms outperform existing approaches. We also present some first mathematical results.

Fault Detection in Multi-Sensor Networks

Francisco Serdio[a], Edwin Lughofer[a]

[a]*Department of Knowledge-Based Mathematical Systems, Johannes Kepler University Linz, Austria*

Abstract

This research is focused in to test a set of different models with fault detection purposes in condition monitoring systems equipped with multi-sensor networks.

Because of the (different) nature of the sensors, a *beforehand* system identification process searches for relations and dependencies between sensor channels measuring the state of system variables, conceptually acting as a fusion operation. Due to the increasing demand of process supervision during the last years, multiple sensor networks turned out to emerge in industrial settings and factories. A manual supervision is not affordable or in some cases simply impossible, because (especially, in large scale systems) the number of sensors installed to collect data for monitoring purposes are often hundreds or a few thousands. Such large scale sensor networks get room for fusioning mechanisms, then playing an intrinsic important role in order to assimilate data in a correct way, also providing an information retrieval source. For a recent survey of widely used (data) fusion methods see [5].

As fusion operators we have used (i) Subset selection and (ii) Channel transformations. Subset selection leads to a combinatorial optimization problem requiring very large computation time, so we have applied a greedy based approach in form of a modified variant of forward selection, leaned on [2], to reduce computation time drastically. Channel transformations, by means of the projection of the complete channel space (including time lags) to a transformed orthogonal space, help to provide a compact view on the whole data set because of out-weighting unnecessary directions in the joint multi-dimensional space. Even when there are different ways in which data can be transformed to be presented as a priori knowledge to a diagnostic system, it is well known that Principal Component Analysis (PCA) [4] and Partial Least Squares (PLS) [3] form a major component of statistical feature extraction methods [6]. We have used both.

Email addresses: francisco.serdio@jku.at (Francisco Serdio), edwin.lughofer@jku.at (Edwin Lughofer)

Afterwards, the identified systems become suitable to be modeled. As modeling techniques, we have explored vectorized time series models and three of its variants, i.e. non-linear finite impulse response models (NFIR), non-linear output error models (NOE) and non-linear Box-Jenkins models (NBJ), which are methods belonging to the class of recurrent models [1]. The vectorized time series models identify k-step ahead multi-dimensional prediction models including the time lags best explaining the target. The non-linear finite impulse response models only rely in the lagged input variables, whereas the non-linear output error models include the lags of the own predictions and the non-linear Box-Jenkins models extend the non-linear output error models by also including the lags of the prediction errors.

Following a previous research line, an unsupervised scheme is used where neither annotated samples nor fault patterns/models need to be available a priori. As before, the identification of the models and the fault detection stage are solely based on the on-line recorded data streams. Testing our approach with four real-world condition monitoring scenarios employing multi-sensor network systems demonstrate that the Receiver Operating Characteristic (ROC) curves are improved in comparison with those ones achieved with models without lags and without transformations (i.e. native static models) by about 20% to 30%.

References

[1] J. Abonyi. *Fuzzy Model Identification for Control*. Birkhäuser, Boston, U.S.A., 2003.

[2] C. Cernuda, E. Lughofer, W. Maerzinger, and J. Kasberger. NIR-based quantification of process parameters in polyetheracrylat (PEA) production using flexible non-linear fuzzy systems. *Chemometrics and Intelligent Laboratory Systems*, 109(1):22–33, 2011.

[3] M. Haenlein and A.M. Kaplan. A beginner's guide to partial least squares (PLS) analysis. *Understanding Statistics*, 3(4):283–297, 2004.

[4] I.T. Jolliffe. *Principal Component Analysis — second edition*. Springer Series in Statistics. Springer, 2002.

[5] B. Khaleghi, A. Khamis, F. O. Karray, and S. N. Razavi. Multisensor data fusion: A review of the state-of-the-art. *Information Fusion*, 14(1):28–44, 2013.

[6] V. Venkatasubramanian, R. Rengaswamy, S.N. Kavuri, and K. Yin. A review of process fault detection and diagnosis: Part iii: Process history based methods. *Computers & Chemical Engineering*, 27(3):327–346, 2003.

s c c h
software competence center
hagenberg

Parallelization of Algorithms for Linear Discrete Optimization using ParaPhrase

author Michael Roßbory and Werner Reisner
Software Competence Center Hagenberg
e-mail *{michael.rossbory,werner.reisner}@scch.at*

Abstract — In industry optimization of processes, production planing, or resource usage is important to reduce costs and increase profit. Mathematical models for optimization can contribute to achieve this, but they also pose some challenges. Not only expertise in mathematics is needed to apply these optimization models, but furthermore expertise in programming is needed for implementation and integration into the software landscape of the company. Additionally most optimization algorithms are computationally very expensive and finding a solution takes a long time. Parallelization reduces the time and can lead to better results, but makes implementation even more challenging. How the high-level pattern-based approach of ParaPhrase and its provided tools reduces this challenges will be described in this paper using a real-world example from industry.

Key words — *paraphrase, high level pattern, parallelization, integer programing*

COMET
Competence Centers for Excellent Technologies
**Funded by the Austrian Government
the state of Upper Austria
the Johannes Kepler University**

Technical Report
SCCH–TR–1331

Parallelization of Algorithms for Linear Discrete Optimization using ParaPhrase

Michael Roßbory
and Werner Reisner
Software Competence Center Hagenberg GmbH
Email: {michael.rossbory,werner.reisner}@scch.at

***Abstract*—In industry optimization of processes, production planing, or resource usage is important to reduce costs and increase profit. Mathematical models for optimization can contribute to achieve this, but they also pose some challenges. Not only expertise in mathematics is needed to apply these optimization models, but furthermore expertise in programming is needed for implementation and integration into the software landscape of the company. Additionally most optimization algorithms are computationally very expensive and finding a solution takes a long time. Parallelization reduces the time and can lead to better results, but makes implementation even more challenging. How the high-level pattern-based approach of ParaPhrase [5] and its provided tools reduces this challenges will be described in this paper using a real-world example from industry.**

I. Introduction

Maximize profit! This is of the main goals of many companies. Optimization based on mathematical models can contribute to this in many ways, e.g. process optimization, optimization of production planing and control, or optimized usage of resources. Optimization in the mathematical sense means finding the minimum or maximum, respectively, of an objective value function that is defined for a certain area, where optimization is required. An example would be minimizing waste when cutting sheets of various dimensions out of big paper coils. Furthermore mathematical optimization can not only find an optimal solution, but can assure that given boundary conditions are fulfilled.

Many different mathematical methods exist to find the optimum of an objective value function under a given set of constraints, but all of them have in common that they are computationally very expensive (many are known to be NP-hard). Parallelizing these methods reduces the computation time and can even lead to better results. Therefore parallelization is a worthwhile goal.

Many optimization methods require heavy mathematical fundamentals, and implementation and integration into the companies infrastructure require a lot of knowledge and experience in programming. Parallelization makes this even more complicated. The **ParaPhrase** approach [5] on parallelization can assist in this. Its high-level pattern-based approach reduces the complexity that low-level parallelization techniques entail and furthermore helps to separate the implementation parts with mathematical focus from the parts with parallelization focus. Therefore the programmer does not need to be expert in both areas.

The **ParaPhrase** provides a set of generic high-level parallel patterns to support the developer to implement efficient parallel applications targeting heterogeneous architectures. The initial patterns comprise a set of *core* patterns: Pipe, Farm, Map, Reduce and a set of *high level*, more domain specific, patterns: Devide and Conquer, Genetic Algorithm that can be used isolated, but can be furthermore arbitrarily nested.

The implementation of this patterns is based on the **FastFlow** framework developed at the University of Pisa [1].

In this paper we want to show how **ParaPhrase** can be applied in discrete optimization problems by means of a real use case directly taken from industry. This use case will be outlined in the next section. After that the applied optimization algorithm and its sequential implementation will be described. The subsequent section will deal with the parallelization of this algorithm and after that first evaluation results will be presented.

II. Use-Case Description

The use case has been taken from a project carried out at the SCCH with a company residing in the area of metal sheet processing. The problem that has to be solved deals with optimization of material consumption to minimize production costs, while certain constraints have to be fulfilled. This kind of problem is widespread and often arises from many applications in industry e.g. in a make-to-order steel company [4]. In pertinent literature, this kind of problem is known as *One-dimensional Cutting-Stock Problem* an optimization problem from the area of discrete linear optimization.

Since the focus of this paper is not a detailed description of this project and the whole the problem specification, we decided to chose a similar example where the same optimization problem has to be solved and where the same optimization algorithm will be used as in the actual project.

The chosen example is about make-to-order reel and sheet cutting in a paper mill. The paper mill has reels with different diameters, widths, and sheet thicknesses and tool sets for sheet cutting with different properties at its disposal to satisfy production orders from customers. The orders specify the number and dimensions of sheets of a given thickness and the number and widths of reels with specific diameters to be produced. The goal is to satisfy the production order while optimizing the use of the tool set and minimizing waste. Use case details and mathematical definitions are provided in [2].

III. Optimization Algorithm

Integer linear programming (ILP) knows many different exact algorithms (e.g. simplex algorithm, inner-point method,

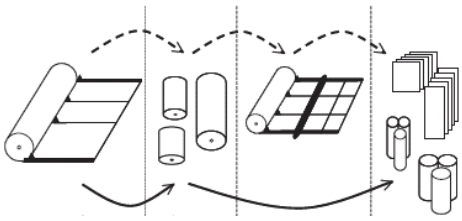

Figure 1. Cutting master reels into sheets and auxiliary reels.

branch-and-bound, branch-and-cut) and heuristic methods (e.g. hill climbing, simulated annealing) to solve optimization problems like this. [6]

The algorithm that has been implemented to solve this use case has a few similarities to Dantzig's *Simplex Algorithm* [3]. Basically this algorithm is two-phase, iterative and stepwise. In the first phase an initial admissible solution is computed, which is needed to start the second phase. Based on this initial solution in every iteration of the second phase with exchange operations it is stepwise tried to find a solution with a better objective function value until an optimal solution is found.

Similar to that algorithm our implementation, too consists of two phases with a first initial phase and an iterative and stepwise second phase.

The first phase again searches for an initial admissible solution that fulfills only one given constraints, applying an algorithm called first fit decrease, and calculates the object function value.

Based on this initial solution found in the first phase, the second phase iteratively tries to find solutions with a better objective function value until the optimum is found or another termination criterion is reached (e.g. maximum number of iterations). In each iteration a chain of steps is traversed, each of these steps heuristically selected from a set of candidate steps. Within this chain an increasing number of constraints is enforced interleaved with optimization of the partial admissible solutions. To pursue different optimization paths (caused by heuristics in the optimization steps) the initial solution is copied several times and pushed into the pool of intermediate solutions. In each iteration an intermediate solution is picked from the pool and passed through the optimization chain. If the new objective function value after passing the chain is better than a certain threshold, the intermediate solution is considered worth for further optimization and again several copies are pushed into the pool. Otherwise the intermediate solution is dismissed.

IV. Sequential Implementation

To solve various optimization problems a generic and problem independent optimization framework had been developed at the SCCH. The described algorithm was the first one that had been implemented within this framework to solve the optimization problem stated above. The framework only contains the algorithm specific classes that are independent from the particular problem that has to be solved as well as base classes on which the problem specific parts, e.g. the actual constraints or optimization steps, base upon. Since only the main components of the generic framework are of interest for the parallel implementation and its evaluation, only those are outlined in the following:

Workpiece: A *Workpiece* corresponds to an intermediate admissible solution. It holds all information about the solution, like its properties or the already performed optimization steps. These objects are passed through the sequence of optimization steps for optimization. Implementations for specific problems base upon this component.

WorkpiecePool: The *WorkpiecePool* holds all the *WorkPieces* and synchronizes access to them.

Worker: A *Worker* equates to one specific optimization step of the optimization chain. Implementations of problem specific optimization steps again base upon this base class.

The initial implementation was sequential and uses a loop to perform the optimization. In each loop iteration one *Workpiece* is taken from the *WorkpiecePool* and passed through the chain of optimization steps. In this implementation the chain corresponds to a sequence of *Workers*, that has been heuristically assembled in the initialization. At the end of each iteration the objective function value for the current *Workpiece* is calculated. Depending on the result the *WorkPiece* is either dismissed if the value is worse than a certain threshold or multiple copies are pushed back into the *WorkpiecePool* for a further optimization iteration.

The framework exposes some important parameters that have an influence on the optimization process and the quality of the solution. Changing this parameters allows controlling the computational workload and are therefore especially important during the evaluation of the sequential and parallel implementation. The parameters that are important for this purpose are:

ObjectNum: After the first phase of the algorithm only one admissible solution is available in the pool. Since the optimization steps include stochastic elements more copies of the initial solution are put into the pool to be able to pursue different optimization strategies. The number of copies is defined by this parameter.

NetLength: This parameter defines the number of optimization steps in the optimization chain.

Iterations: Since the number of iterations to find the optimal solution might be very high, an upper limit can be defined with this parameter.

V. Porting

This section describes the approach to parallelize the sequential implementation using the parallel patterns that are available in **ParaPhrase** so far. Furthermore some extensions and variations of this approach are outlined that may lead to additional performance improvements.

A. Pipeline approach

Since the algorithm is stepwise and iterative, and the chain of optimization steps, though heuristically assembled, stays the same in every iteration, parallelization using the *pipeline* pattern is most obvious. In this approach the pipeline equates to the optimization chain where a single stage of the pipeline corresponds to a specific optimization step. Additionally to the optimization stages a last pipeline stage is added that performs the calculation of the objective function value and decides

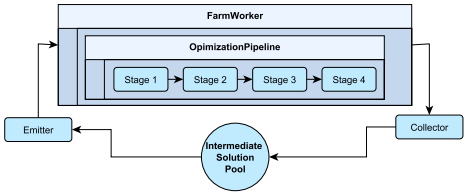

Figure 2. Current parallel implementation using an outer farm with emitter, collector and one worker pipeline.

how the intermediate solution is further processed (dismissed or pushed back to the pool). The outer loop that takes the intermediate solutions from the pool and sends them through the pipeline is the same as in the sequential implementation.

B. Farm approach

Basic concepts of the optimization framework are problem independence and extensibility. To foster this concepts also in parallelization the single pipeline approach has been extended. Therefore the optimization pipeline has been nested in a *farm* pattern as *farm worker* (Figure 2).

The components of the farm are the following:

Emitter: The emitter of the farm has access to the intermediate solution pool. It takes the intermediate solutions from the pools and emits them to the worker(s). This replaces the loop in the sequential implementation that took the solutions from the pool and passed them through the optimization steps. Like the replaced loop the emitter takes the solutions from the pool until a solution is found, the pool is empty or another termination criterion is reached.

Worker: A worker of the farm contains a nested pipeline that again corresponds to the chain of optimization steps. This time only the optimization steps are included in the pipeline without an additional collector stage.

Collector: After an intermediate solution has passed through the pipeline the collector of the farm is responsible for the calculation of the objective function value for this solution and the decision how the solution is further processed.

C. Future extensions

The use of a farm as the outer pattern with nested optimization pipelines has various advantages compared to the single pipeline approach, especially concerning performance and extensibility.

- If enough cores are available on the target machine, more farm workers containing the same pipeline can be used which would lead to additional performance improvements. But when using multiple pipelines, the collector might become a bottleneck, because it has to evaluate every intermediate solution from every pipeline. Therefore it might be better to omit the farm collector and add a kind of collector as the last stage to every pipeline. But at least the intermediate solution pool is shared by all these collectors and access has to be synchronized. Splitting the pool into different storage regions for every collector and only synchronizing the access of the emitter might lessen the synchronization overhead and improve the performance further. (Figure 3)
- Variations of the current algorithm might use multiple chains of optimization steps that differ in order or selection of the steps. With an outer farm the implementation would just require an appropriate *loadbalancer* implementation. The loadbalancer is used by the emitter to decide to which worker an item has to be passed. The default strategy in **ParaPhrase** is Round Robin. But other strategies can be implemented where the decision is based on some properties the item provides. (Figure 4)
- In the paper mill optimization problem the execution times for the single optimization steps are almost equal. But other problem statements might contain optimization steps with much longer execution times compared to the other steps. To compensate this the pipeline stages with the long running steps just have to be replaced with nested farms. The longer the step takes the more workers have to be added to the nested farm.

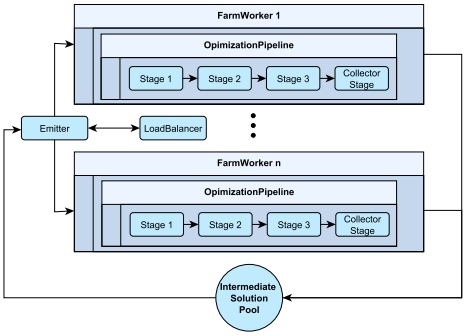

Figure 3. Parallelization with an arbitrary number of worker pipelines with a collector stage attached to each pipeline.

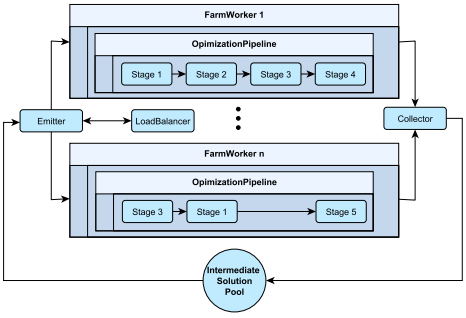

Figure 4. Parallelization with different worker pipelines and a loadbalancer for distributing the workpieces to the correct pipeline.

VI. Evaluation

Evaluation has been performed on different hardware platforms ranging from rather low-performance quad-core machines to high-performance clusters. Performance gain has been achieved on all those platforms. The following evaluation has been performed on a 12 core machine (dual-hexacore, Intel Xeon X5690, 3.47 GHz) with HyperThreading support, and 24GB of memory, running under Linux 64bit (Ubuntu 12.04), to show the performance gain that can be achieved even on quite low cost systems. The specifications of the evaluation

system is important because they seem to have a strong relation to the results and its interpretation.

To be able to evaluate and compare the different approaches the source code has been instrumented with additional statements to measure the overall execution time of one run of the program. The measurement is started after the configuration files (that contain the problem specifications) have been read and stopped after a solution has been found (For this test case the parameters were chosen in a way that the problem has a solution). The measurements are not constrained to the parallelized, or sequential parts, respectively, containing the code were the actual iterative optimization takes place. The initialization parts, that are different between the sequential and the parallel implementation, are also timed to include the initialization overhead that the different approaches have. For measuring the execution time, the timer class provided by **FastFlow** has been used. The measures are in microseconds and converted into seconds with two decimals afterwards. The denoted values are averaged values over three runs (whereat irreproducible outliers have been excluded).

With the results of the evaluation runs we want to analyze the approaches according to:

- Changes of the behavior of the parallel and sequential algorithm, respectively, using different value combinations of the previously mentioned parameters *NetLength* and *ObjectNum*. to achieve different numbers of threads and variable computational workload.
- The differences of the sequential and parallel version, especially regarding to the achieved speed up.

The *first evaluation step* was to determine the number of instantiated threads, the workload and the number of used CPUs, caused by the selected architecture of used parallel patterns, depending on the selected parameters described above. The observed result was that the number of threads always equates to the value of *NetLength* plus three. The value of *NetLength* determines the number of pipeline stages, where each stage corresponds to one thread. The additional three threads comprise of the main thread of the program and the threads for emitter and collector of the used farm. During program execution the number of fully utilized cores was always two less than the number of threads. The reason therefore is that the main thread is just waiting for the farm to finish and does not use CPU time and that the emitter has nearly no computations to perform and mostly just waits for objects to arrive in the pool (no busy-waiting). On the whole, this is exactly the expected behavior.

In the *second step*, the execution time was recorded with a *NetLength* from 15 to 20. With each *NetLength* multiple runs with different values (from 25 to 100 with a step size of 25) for *ObjectNum* were performed. As expected the increase of the value for *ObjectNum* leads to higher execution times in the sequential as well as in the parallel version. The same is true for increased *NetLength* values. Calculating the resulting speed up achieved by parallelization showed that the higher the value for *ObjectNum* was, the greater was the speed up, ranging from about 5 to over 7 times.

The intention of the *third step* was to evaluate the execution time and speed up with values for NetLength ranging from 8 to 40, but leaving the ObjectNum at a constant value of 100.

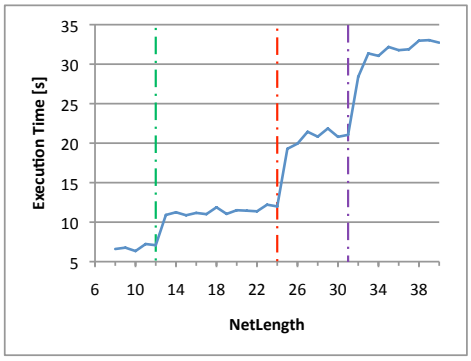

Figure 5. Evolution of execution time of parallel optimization with increasing number of optimization steps.

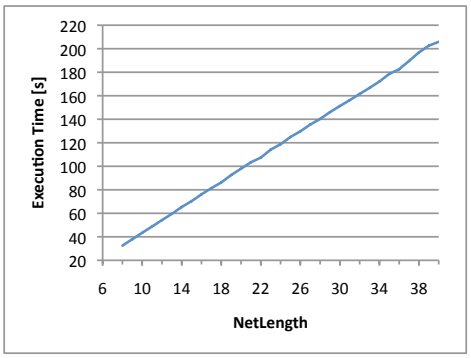

Figure 6. Evolution of execution time of sequential optimization with increasing number of optimization steps.

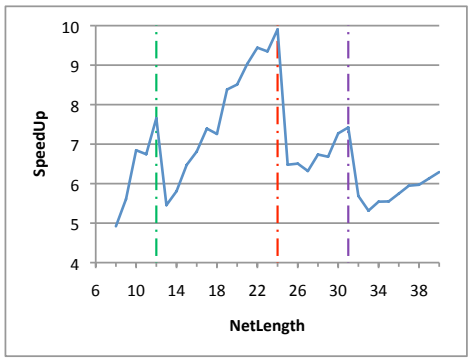

Figure 7. Speed up of parallel implementation compared to the sequential implementation with increasing number of optimization steps.

This range has been chosen regarding to the specifications of the evaluation hardware platform (12 physical cores with hyper threading). This interval leads to a range of threads, starting with a number of threads less than the available physical cores and ending with a number of threads much higher than physical and logical cores (hyper threading support). This way all important ranges and transitions of thread numbers are included in the evaluation: Threads less than or equal to number of physical cores [8; 12], threads within the range of available physical and logical cores [12; 24], threads higher than the number of all cores, and the transitions between those ranges. The results are visualized in the charts in Figures 5, 6 and 7.

The evolution of the execution times of the sequential implementation is shown in Figure 6. As expected the increase of the execution times is linear. The reasons for the linear increase are that in every run the number of steps in the optimization chain was increased by 1 and all available optimization steps need about the same time for computation. If the execution times of the single steps would be very different (as they might in future scenarios) and since the steps are selected randomly, the values would still increase quite linear, but not as exact as in this example.

More interesting are the results of the parallel implementation, shown in Figure 5. As expected the execution times increase as the number of optimization steps do, but this time not linearly. In the range of 8 to 12 for the NetLength parameter the execution time is almost constant, most likely because there are still unused physical CPUs. At the NetLength of 13 there is jump in the measured execution time (indicated by the green dashed line). The reason for this seem be that there are no unused physical CPUs left and that from this point on the logical CPUs (available due to hyper threading support) have to be used (This might be a hint to the hyper threading aware scheduling of the Linux kernel).

This observed developing of the speedup rate meshes with the default pinning strategy of **FastFlow**, where each worker is pinned to an unused physical CPU. More workers than CPUs leads to more than one worker pinned to the same physical CPU. As a consequence CPUs that have less workers and have already finished their work can not take over work from CPU with more workers and therefore execution time suddenly increases.

Until a NetLength of 24 the execution time is again almost constant, due to the available logical CPUs that were unused so far. At 25 there is again a jump (indicated by the red dashed line), but this time a greater one. The reason therefore is most likely that now there are more optimization steps (and therefore more threads) than CPUs and that the CPUs now have to be shared among the threads.

So far the observed behavior corresponds to what has been expected. Therefore increasing the NetLength further, it was expected that the execution time again stays almost constant with jumps at multiple values of physical and logical CPUs. But what has been observed was a great jump (Indicated by the purple dashed line) at a NetLength of 31. So far a reasonable explanation therefore could not be found.

VII. Conclusion

On the whole, the implementation of the specific parts containing the code for the actual parallelization took much less time than understanding and refactoring the existing sequential implementation in order to be able to parallelize it at all regardless of which approach scheme would be used. The most time-consuming problems were synchronizing the access to shared resources (in emitter and collector), identifying the objects that in the sequential implementation were only instantiated once, but multiple times in the parallelized version and implementing the required copy constructors and assignment operators.

It is interesting how far the refactoring tools of **ParaPhrase** of will support the developer in this in future, and how much the effort will be reduced compared to manual refactoring.

Dynamic adjustment of the parallelization (e.g. used pattern hierarchy) depending on runtime behavior will also pose a challenge. An already mentioned example is the possible need for a nested farm within the optimization pipeline to replace a long running optimization stage. Whether there are stages that need much more time for execution, or how much longer that time it is, might not be known before the first run. This kind of dynamic problem or execution time depended pattern architecture might be a harder problem to solve, but would be of much benefit to this use case. Looking at the evaluation results concerning to speed up and scaling in comparison to time and effort that had to be spent to achieve them, it is obvious that it is worth to put time in. Even when using **ParaPhrase** for the first time and with little experience in parallelization at all. More practice will reduce the effort for parallelization even more, because you get used how to apply the parallel patterns and to avoid common pitfalls. This effect of reduced time due to more practice seems greater using **ParaPhrase** compared to low level approaches like *OpenMP*, because the high-level pattern-based approach is not as problem and implementation depended as low level approaches are. It furthermore has to be noted that **ParaPhrase** does not yet offer all planned features and tools that will support the developer during the parallelization process, e.g. tools for refactoring. Availability of those tools will make parallelization even more easier and faster.

A last point that has to be mentioned concerns extensibility and variability. The optimization framework is still in an early stage of development. New algorithms will be added and existing ones will have to be changed due to new requirements. The pattern-based approach makes a proper adaption of the parallelization easier and less expensive than it would be using low-level parallelization strategies.

References

[1] Marco Aldinucci, Marco Danelutto, Peter Kilpatrick, and Massimo Torquati. Fastflow: high-level and efficient streaming on multi-core. In Sabri Pllana and Fatos Xhafa, editors, *Programming Multi-core and Many-core Computing Systems*, Parallel and Distributed Computing, chapter 13. Wiley, January 2013.

[2] M.Helena Correia, Jose F. Oliveira, and J.Soeiro Ferreira. Reel and sheet cutting at a paper mill. *Computers and Operations Research*, 31(8):1223 – 1243, 2004.

[3] G.B. Dantzig and M.N. Thapa. *Linear Programming: 1: Introduction*. Linear Programming. Springer, 1997.

[4] J.M. Valério de Carvalho and A.J. Guimarães Rodrigues. An lp-based approach to a two-stage cutting stock problem. *European Journal of Operational Research*, 84(3):580 – 589, 1995. Cutting and Packing.

[5] Kevin Hammond, Marco Aldinucci, Christopher Brown, Francesco Cesarini, Marco Danelutto, Horacio González-Vélez, Peter Kilpatrick, Rainer Keller, Michael Rossbory, and Gilad Shainer. The paraphrase project: Parallel patterns for adaptive heterogeneous multicore systems. In Bernhard Beckert, Ferruccio Damiani, FrankS. Boer, and MarcelloM. Bonsangue, editors, *Formal Methods for Components and Objects*, volume 7542 of *Lecture Notes in Computer Science*, pages 218–236. Springer Berlin Heidelberg, 2013.

[6] Frederick S. Hillier, Gerald J. Lieberman, Frederick Hillier, and Gerald Lieberman. *MP Introduction to Operations Research*. McGraw-Hill Science/Engineering/Math, 2004.