A Grid-enabled Framework for Exact Optimization Algorithms

I. Zunino, N. Melab and E-G. Talbi
Laboratoire d'Informatique Fondamentale de Lille
Université des Sciences et Technologies de Lille
59655 - Villeneuve d'Ascq Cedex - France
E-mail: izunino@exa.unicen.edu.ar; {melab,talbi}@lifl.fr

Abstract

In this paper we present a framework for writing exact optimization algorithms distributed on a grid environment. It presents a new way of reusing design and code for multi-objective optimization methods in conjunction with assistant methods. These kinds of methods are used mainly for reducing the search space, or for using a mono-objective method for solving a multi-objective problem, or both. We use a master-slave paradigm for the parallelization of the work units and a branch and bound algorithm as a default assistant method. The branch and bound algorithm is also distributed on grids which allows a two level parallelism for the optimization. We show how the different objects are codified in order to allow less communication while at the same time maintaining the reusability requirement. A sample instantiation of the framework is presented using the Parallel Partitioning Method (PPM). Preliminary results are shown using different Flowshop instances.

Keywords: *Frameworks, Branch and Bound, Parallel Computing, Grid Computing, Multi-Objective Optimization, Flow-Shop Problem*

1 Introduction

There exist a grand variety of frameworks for exact optimization. Most of these frameworks use a branch and bound algorithm to perform the search, or facilitate the programmer in writing branch and bound-based algorithms. Examples of such frameworks are: PUBB [19], BOB++ [2], PPBB [16], PICO [7], MALLBA [4], ZRAM [1], ALPS [20], MW Framework [8], Symphony [14]. There is a good taxonomy of parallel software frameworks and an overview of the implementation of parallel branch and bound algorithms and frameworks in [15].

In multi-objective optimization there exist different methods or strategies to optimize without searching in all the search space. Examples of methods which use those strategies are the Two Phases Method (TPM)[17], the Parallel Partitioning Method (PPM)[9] and the epsilon-constrained method[18]. These methods have the particularity of optimizing using a strategy. This strategy is used, in order to reduce the search space, for using a mono-objective method for solving the multi-objective problem, or both. In order to do this, it uses another method (an assistant method) to optimize different subspaces. Hence, each time one of these methods is written, a new implementation of the assistant method has to be written or at least, the connection between the strategy and the method has to be built. At the moment, to the best of our knowledge, there are no distributed frameworks which address directly the use of methods for multi-objective optimization by the assistance of another method. Our framework addresses these kinds of problems using a default assistant method and at the same time being distributed on a grid environment. In our work, we will be having a default implementation using as an assistant method the branch and bound algorithm developed in [12]. This algorithm can be used with different problems, and accepts multi-objective and mono-objective optimization. It has proven to be very efficient and highly scalable on a grid environment [12].

The rest of the paper is organized as follows: Section 2 presents the design requirements and objectives and the overall architecture of the proposed framework from design and implementation points of view. Section 3 describes the application of the framework for solving the bi-objective Flow-Shop scheduling problem and its experimentation on the Grid5000 French experimental grid. In Section 4, the conclusion and perspectives of the work are drawn.

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

2 The Framework

Usually, a family of related applications has many of their functionality similar, if not the same. A framework addresses this aspect by providing abstract representations of classes and implements invariant parts for the different applications in a family. More specifically, "A framework is a set of abstract classes and components that together comprises an abstract design solution for a family of related applications" [10].

2.1 Design requirements and objectives

A framework lets us make different applications within a domain of functionality. In order to be able to do that, it is needed that we specifically define the domain. The functionality the framework is expected to satisfy is called functional requirements. There are also quality criteria which the framework should try to satisfy and they are called non-functional requirements.

Functional requirements

- The framework should facilitate the implementation of different exact optimization methods for exploring the search space.
- There should provide parallel support for the different parallel exact optimization methods that need it such as TPM or PPM.
- It should be able to distribute the provided parallelism on grids.

Non-functional requirements

- Reusability: the code and design solutions should be reusable.
- Modifiability: the development of new methods should be done in a clear way concerning the underlying optimization method assistant and parallel behavior. That is, the programmer should have to write only the part of the new method.
- Flexibility: the framework should be easily extended to provide the programmer with the ability to easily instantiate different optimization domains. That is, using different objectives, restrictions or models for the optimization.
- Extensibility: It should be easily extended in order to support new parallel or distributed technologies without the need of changing the code for the optimization algorithms or search strategies.
- Performance: it should provide by default with high-performance optimization algorithms so that the users can focus only on the details concerning their new search strategy for optimization. The distributed and parallel requirements should be met without degrading the performance of the system.
- Scalability: As the framework should be distributed on a grid environment where a large number of processors may be available, it is important that the performance is improved as the number of processors used grows.

2.2 System Architecture

In order to satisfy the requirements above, we implement different design decisions going from an architecture point of view to a design point of view. The architecture lets us address most of the functional and non functional requirements by providing a general view of the interactions of the different modules inside the framework without the need of a detailed specification. The design lets us see more in detail how the framework will be used and instantiated.

Concerning the goals for modifiability and extensibility, we think that working with a layers style makes a strong separation of functionality so that the change in one of the main features would conclude in changing as less code as possible or nothing at all for the other layers. Concerning the functional requirement for parallel search algorithms, we provide with a layer that implements the functions for optimization using asynchronous communication so that each call to those functions doesn't stop the flow of the main structure of the optimization method. The goal of distributing the work on a grid environment is addressed using components with distributed capabilities, that is, they should be serialized (codified) and deserialized (decodified) in order to be sent through the grid. At the same time, with the purpose of making the framework as less attached as possible to any technology, this is implemented in a different layer. The framework provides with a default implementation of this layer and we think making it work with the Message Passign Interface (MPI) is a good option since the programs can be deployed on a cluster or on a grid environment.

The first design decision we take is the separation of the main method for exploring in the search space (i.e.: TPM, epsilon-constrained, PPM) from the solver that performs the optimization within a particular space (i.e. B&B). The StrategySolver layer has the responsibility of optimizing the space in which to explore for the solutions. In order to find these solutions, it uses the OptimizationSolver layer which responsibility is the optimization of an objective function inside a given space. In this way, each time a new

algorithm is written, there is no need to make any changes to the optimization code.

As it was previously said, the StrategySolver should be able to work in parallel for some of its functions. This is achieved by making the OptimizationSolver able to receive and work with asynchronous calls. So, each time the StrategySolver has to perform different optimization tasks in parallel, it calls the OptimizationSolver as many times as needed.

Concerning the distribution of the paralellism there is another layer that performs the distributed calls so that the optimization layer may take place in different machines. By default, this is done using the MPI layer. The architectural view is represented in Figure 1. In this view all the layers of the MPI API are not shown and depend on the implementation. However, as it was previously said in the extensibility requirement, this layer can be changed without the need of changing the implementation of either the strategy or the optimization algorithm used, as long as it conforms to the same interfaces.

A Sequence Diagram used to show the actions used during the distribution of optimization requests to the different slaves is shown in Figure 2. In this example, we can see the strategy (StrategySolver) calls asynchronously (the flow of the method is not interrupted) the master (MasterMPISolver) two times to optimize . At each time, the MasterMPISolver calls one different Slave and after that it remains waiting for the answer in the method getSolutions(). Then, each of the slaves call its own solver which would actually perform the optimization. After the internal solvers finish optimizing, each slave sends the results to the Master. When all the results arrive the master returns the answers to the strategy method. In this Figure, in order to make it readable, we hide the part of the codification of the objects that where distributed. The codification will be shown later.

2.3 Grid-based implementation

A summary of the main classes used for the optimization process is represented in Figure 3. The class OptimizationSolver is the abstract class used to represent the main method and the assistant method (i.e. the branch and bound). This solver has a method to perform optimization with Objectives, Restrictions and InitialSolutions. Each of these components is represented with a different class. The Objective class has a method which returns the evaluation of that solution for a specified objective. The Restriction class has also an evaluation method and it also works with solutions, but returns a boolean value indicating whether the solution is inside the feasible space or not. The initialSolutions can be used to feed the solver or the strategy with initial solutions for optimizing the search. This is also used in some of the strategies as we shall see later in the example of PPM.

For the distribution part, we implement a wrapper for the OptimizationSolver which acts as a proxy, redirecting the different asynchronous calls to different machines. This wrapper is represented in both classes: MasterMPISolver and SlaveMPISolver. They use a master-worker paradigm to distribute the work. The Master is connected to the main method or strategy and the Solver is connected to the assistant method. In our default implementation using the branch and bound proposed in [12], the assistant method is also distributed on the grid so this generates a two level paralellism and distribution which we believe can be helpful in some of the optimization methods. The fact that the optimization method, the assistant method and the distributed capabilities use the same interface will let us compose different types of interactions between them to achieve different results, through a composition of strategies and methods.

In order for the different classes to be distributed, they have to be codified and decodified. Each class that is distributed codifies and decodifies itself.Depending on the codification used, this may greatly optimize the transfer of information since only the necessary details of each of the objects would be transmitted. The way to codify and decodify each object is implemented using the basic types used in c++. Each codification is made using a Packet object which will later be distributed. The Packet class lets us hide the distributed technology used to codify the objects. This is done in order to separate the distributed technology from the rest of the framework (Figure 4). In this way, reusing the codification of the objects when changing the distributed technology would conclude in no modification to the serializable classes. In our default implementation of the framework there exist the MPIPacket class that implements its methods.

As an example for the codification, each bound restriction is represented by an objective (in this case as an integer) and a lower and an upper bound. Therefore, each time a BoundRestriction is sent, we would only have to send one integer and two double values. This should greatly optimize data transfers since only the necessary data is sent.

Finally, in order to use the framework on a grid environment, we use MPICH-G2 [13], a grid-enabled implementation of MPI which uses the grid services provided by the Globus Toolkit. In this way, we can execute our framework in different clusters with a wide range of heterogeneus pro-

Figure 1. Architectural view using the distributed implementation of MPI

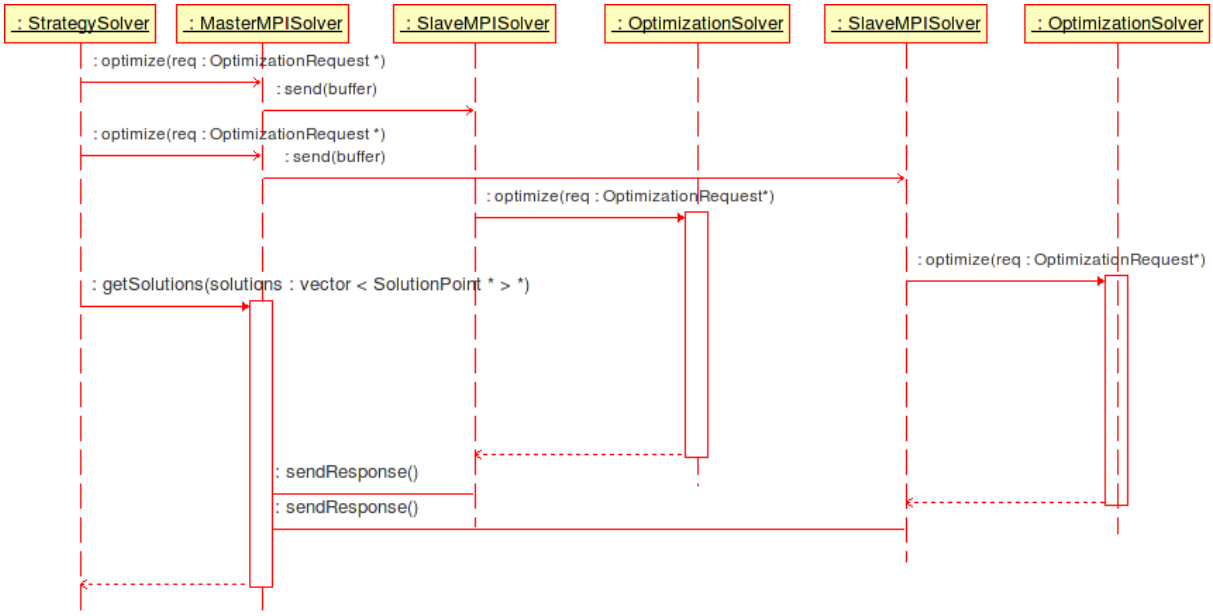

Figure 2. Sequence Diagram showing the distribution of requests

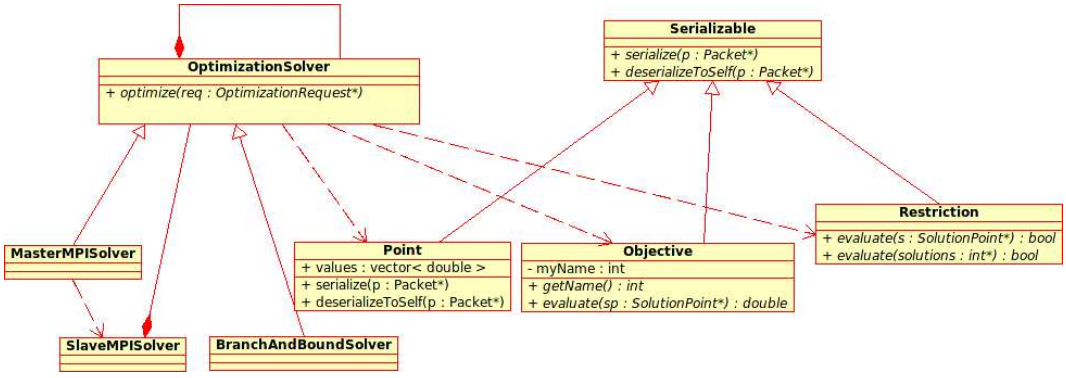

Figure 3. Classes used during the optimization process

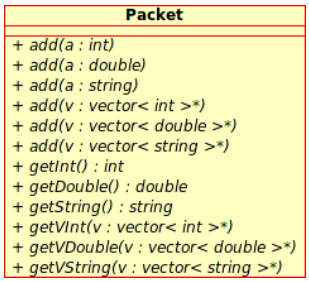

Figure 4. Packet class used for the codification

cessors.

3 Application to the Bi-objective Flow-Shop Problem

3.1 Flow-Shop problem formulation

The Flow-Shop problem is one of the numerous scheduling problems [3]. It has been widely studied in the literature. The problem consists in scheduling n jobs (i = 1 .. n) on m machines (j = 1 .. m). In this paper, we focus on the permutation Flow-Shop where the jobs are scheduled in the same order in all the machines. The two considered objectives are the makespan (Cmax) and the total tardiness (T). The makespan is the completion time of the last job and the total tardiness is the sum of the tardiness of every job. In the Graham *et al.* notation [5] this problem is denoted F/Permut, di/(Cmax,T).
The makespan minimization problem has been proved to be strongly NP-hard by Garey, Johnson and Sehti [11] for permutation Flow-Shops with more than two machines whereas the total tardiness minimization problem has been proved to be NP-hard by Du and Leung [6] even on a single machine.

3.2 The framework instantiation on the problem using the PPM strategy

The PPM strategy consists of three phases which are represented in Figure 5. In this example, the first phase finds the extremes of the two objectives. During the second phase, the search space is uniformly divided with respect to one extreme. Each subspace is optimized with respect to the other extreme. Then, we have a set of uniformly located solutions. The third phase finds the remaining Pareto solutions in all the search space. Using the solutions previously found, as shown in phase three of Figure 5, the space of the multiobjective problem is reduced. The

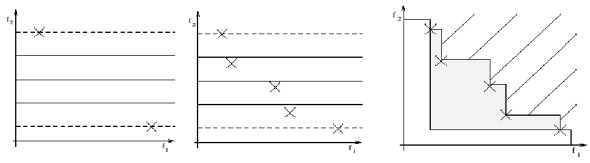

Figure 5. PPM Phases for a bi-objective problem

complete explanation and rationale behind the different steps of PPM can be found in [9].

In order to use the framework for this problem and this method of optimization, one has to implement the classes and methods associated to the strategy, the restrictions used, the assistant method and the objectives of the problem. The classes needed for the instantiation are:

- PPMSolver: It represents the PPM method. In the method solve(OptimizationRequest) we implement the three phases described above. Inside each of these phases, the optimization method is called, for example, to solve one extreme.

- FlowshopObjective: It represents both objectives Tardiness and Makespan. We define the method solve(Solution) which, depending on the kind of objective will evaluate such objective with the solution given. Besides implementing the Objective interface it has to implement bound_abstract. This interface is used by the branch and bound each time it calculates the bounds.

- BoundRestriction: It defines constraints for the 2nd stage of PPM, which several searches are sent in a different search space. Here we define a lower bound and an upper bound and the method that evaluates the solutions between those bounds. The objects of this class will be used by the branch and bound before inserting them into the Pareto front.

- main function: In this function we initialize the framework, create the different classes and we configure them (composition) to work together. That is, we configure the new PPMMSolver with the MasterMPISolver as the internal solver and the branch and bound solver as the internal solver of the SlaveMPISolver.

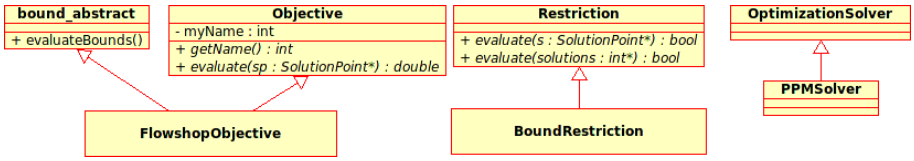

Figure 6. Instantiation of the framework with PPM and the Flow-Shop problem

3.3 Experimentation on Grid5000

Experiments were conducted on the Grid5000 grid. Grid5000 is a nation-wide experimental grid composed by 9 clusters distributed over several French universities (Bordeaux, Lille, Rennes, Sophia-Antipolis, Toulouse, Orsay, Lyon, Grenoble and Nancy). We used the cluster hosted at the Rennes site for the experiments. All of the machines are bi-processors.

Table 1 shows results from different flowshop instances. Inside we show the number of processors used during the different runs, the comparison of the phases of PPM, the total amount of time taken for the resolution, the speed-up of each experiment and the parallel efficiency. We have performed experiments using the instances of Taillard and Reeves. The results show a speed-up that is almost linear with respect to the number of processors. Hence, we think the scalability requirement is met. However, more experiments have to be conducted on different instances and different problems in order to support our initial results.

4 Conclusions and future work

In this paper we have presented an overview of an object-oriented framework for mono-objective and multi-objective exact optimization. As far as we know, there are no frameworks that address the issues of optimizing using a strategy above other methods. We have seen the architecture of the system and the process used for combining the main method or strategy with the assistant method. The codification of the units provided shows a way of codifiyng the objects prior to distributing them taking only the necessary data needed for their identification. Its object oriented nature lets us change the distributed technology without changing the objects codified.
An application using the PPM strategy and a flowshop problem was presented. Through this example we showed the reusability of the framework by implementing only the parts that are needed for the optimization and leaving the rest (like the distribution of requests) for the framework to do. Through numerous experiments with this instance we have proven the scalability requirement is met on a different number of processors.

However, we think that we still have to face different concerns. Specifically, the framework has to be tested with other kinds of problems (TSP, QAP, Knackspack) and other kinds of strategies (K-PPM, TPM, e-contrained). Another issue which needs to be addressed due to the volatile nature of the grid, is fault tolerance. We think this problem could be overcome with different versions of MPI implementations like OpenMPI, FTMPI, MPICH-V2, but we still have to test them with the framework working on the grid. It would be also interesting to integrate this framework with software for heuristic or meta-heuristic optimization.As the interface of the optimization method accepts initial solutions, at least a high level of cooperation is feasible. That is, an heuristic can be used to feed the exact method with initial results. This cooperation will allow the framework to be used for solving larger and more difficult problems.

References

[1] K. Fukuda A. Brunnger, A. Marzetta and J. Nievergelt. The parallel search bench zram and its applications. *Annals of Operations Research*, 90, 1999.

[2] V. D. Cung A. Djerra, B. Le Cun and C. Roucairol. Bob++: Framework for solving optimization problems with branch and bound methods. In *High Performance Distributed Computing*, 2006.

[3] S. Heragu A. Nagar, J. Haddock. Multiple and bicriteria scheduling: A literature survey. *European Journal of Operational Research*, 81:88–104, 1995.

[4] E. Alba, F. Almeida, M. Blesa, C. Cotta, M. Díaz, I. Dorta, J. Gabarró, J. González, C. León, L. Moreno, J. Petit, J. Roda, A. Rojas, and F. Xhafa. Mallba: A library of skeletons for combinatorial optimization. In B. Monien and R. Feldman, editors, *Euro-Par 2002 Parallel Processing*, volume 2400 of *Lecture Notes in Computer Science*, pages 927–932. Springer-Verlag, Berlin Heidelberg, 2002.

[5] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey.

NProcs	Instance	Phase 1	Phase 2	Phase 3	Total	Speed-up	Par. eff.
1	Ta 20-10-1	98min 37s	11m41s	509m23s	619m41s	1	1
60	Ta 20-10-1	3min 6s	45s	9min 24s	13min 43s	45.18	0.75
90	Ta 20-10-1	2min 27s	37s	5min 51s	10min 37s	58.35	0.65
180	Ta 20-10-1	1min 49s	31s	3min 57s	6m 20s	102.71	0.57
1	Ta 20-10-2	159m 41s	23m49s	487m36s	671m8s	1	1
60	Ta 20-10-2	6m 34s	1m30s	14m25s	22m37s	29.67	0.49
90	Ta 20-10-2	4m37s	1m33s	10m13s	16m31s	40.63	0.45
180	Ta 20-10-2	2m57s	1m9s	5m9s	9m20s	74.29	0.41
1	Ta 20-10-3	336m32s	17m51s	442m	795m40s	1	1
60	Ta 20-10-3	6m9s	1m15s	11m17s	19m11s	41.45	0.69
90	Ta 20-10-3	4m32s	51s	8m20s	13m45s	57.82	0.64
180	Ta 20-10-3	2m39s	40s	4m24s	7m47s	102.15	0.57
1	Reeves Easy 20-15-13	104m28s	60m15s	63m40s	228m25s	1	1
60	Reeves Easy 20-15-13	6m32s	2m36s	1m49s	11m14s	20.33	0.34
90	Reeves Easy 20-15-13	5m50s	2m5s	1m33s	9m37s	23.75	0.26
180	Reeves Easy 20-15-13	4m11s	1m24s	1m5s	6m47s	33.67	0.19
1	Reeves Easy 20-15-17	305m29s	51m17s	293m25s	650m12s	1	1
60	Reeves Easy 20-15-17	6m13s	3m37s	5m22s	15m47s	41.20	0.68
90	Reeves Easy 20-15-17	6m39s	1m6s	3m56s	12m5s	53.80	0.60
180	Reeves Easy 20-15-17	4m35s	47s	2m21s	7m54s	82.30	0.46

Table 1. Initial results with PPM and a bi-objective Flowshop Problem

In *Annals of Discrete Mathematics*, volume 5, pages 287–326. 1979.

[6] Du J. and Leung J. Y.-T. Minimizing Total Tardiness on One Machine is NP-hard. *Mathematics of operations research*, 15:483–495, 1990.

[7] C.A. Phillips J. Eckstein and W.E. Hart. Pico: An object oriented framework for parallel branch-and-bound. Technical report, RUTCOR Research Report, 2000.

[8] J. Linderoth J. Goux and M. Yoder. Metacomputing and the master-worker paradigm. Technical report, Angcnne National Laboratory, 1999.

[9] C. Dhaenens J. Lemesre and E-G Talbi. Parallel partitioning method(ppm): A new exact method to solve multi-objective combinatorial problems. *Comput. Op. Res.*, 2006.

[10] Ralph E. Johnson. Documenting frameworks using patterns. In Andreas Paepcke, editor, *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, volume 27, pages 63–72, New York, NY, 1992. ACM Press.

[11] R. Sethi M. Garey, D. Johnson. The complexity of flowshop and jobshop scheduling. *Mathematics of Operational Research*, 1:117–129, 1976.

[12] M. Mezmaz, N. Melab, and E-G. Talbi. A Grid-enabled Branch and Bound Algorithm for Solving Challenging Combinatorial Optimization Problems. In 21^{th} *IEEE Intl. Parallel and Distributed Processing Symp., Long Beach, California*, Mar. 26-30 2007.

[13] B. Toonen N. Karonis and I. Foster. Mpich-g2: A grid-enabled implementation of the message passing interface. *Journal of Parallel and Distributed Computing (JPDC)*, 63:551–563, May 2003.

[14] T.K. Ralphs and M. Guzelsoy. The symphony callable library for mixed-integer programming. In *The Proceedings of the Ninth INFORMS Computing Society Conference*, 2005.

[15] B. Le Cun T.G. Crainic and C. Roucairol. *Parallel Combinatorial Optimization*, chapter Parallel Branch-and-Bound algorithms. Wiley, John & Sons incorporated, 2006.

[16] S. Tschoke and T. Polzer. Portable branch and bound library user manual library version 2.0. Technical report, University of Padeborn, 1998.

[17] E.L. Ulungu and J. Teghem. The two phases method: An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, 20:148–165, 1995.

[18] L. Ladson Y. Haimes and D. Wismer. On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Transactions on System, Man and Cybernetic*, 1:296–297, 1971.

[19] M. Higaki Y. Shinano and R. Hirabayashi. A generalized utility for parallel branch and bound algorithms. In *Symposium on Parallel and Distributed Processing*, pages 858–865, 1995.

[20] L. Ladanyi Y. Xu, T.K. Ralphs and M.J.Saltzman. Alps: A framework for implementing parallel search algorithms. In *The Proceedings of the Ninth INFORMS Computing Society Conference*, 2005.