A Rapid Heuristic for Scheduling Non-Preemptive Dependent Periodic Tasks onto Multiprocessor

Omar Kermia, Yves Sorel

A Rapid Heuristic for Scheduling Non-Preemptive Dependent Periodic Tasks onto Multiprocessor

Omar Kermia, Yves Sorel

INRIA Rocquencourt,
BP 105 - 78153 Le Chesnay Cedex, France
Phone: +33 1 39 63 52 60 - Fax: +33 1 39 63 51 93
omar.kermia@inria.fr, yves.sorel@inria.fr

Abstract

We address distributed real-time applications represented by systems of non-preemptive dependent periodic tasks. This system is described by an acyclic directed graph. Because the distribution and the scheduling of these tasks onto a multiprocessor is an NP-hard problem we propose a greedy heuristic to solve it. Our heuristic sequences three algorithms: assignment, unrolling, and scheduling. The tasks of the same, or multiple, periods are assigned to the same processor according to a mixed sort. Then, the initial graph of tasks is unrolled, i.e. each task is repeated according to the ratio between its period and the least common multiple of all periods of tasks. Finally, the tasks of the unrolled graph are distributed and scheduled onto the processors where they have been assigned. Then, we give the complexity of this heuristic, and we illustrate it with an example. A performance analysis comparing our heuristic with an optimal Branch and Cut algorithm concludes that our heuristic is effective in terms of scheduling success ratio and speed.

1 Introduction

Distributed real-time embedded applications found in domains such as avionics, automobiles, autonomous robotics, and telecommunications, etc, lead to non-preemptive multiprocessor scheduling problem with precedence and strict periodicity constraints. These applications are of crucial importance because if all the constraints are not satisfied this may have disastrous consequences. On the other hand they must minimize their total execution time in order to decrease the delay in the feedback control occurring in these applications.

The problem of non-preemptive scheduling periodic tasks on a multiprocessor architecture was early discussed by Liu in 1969 [1]. In this general problem a task may use either zero or one resource at a time. To solve this problem Baruah [2] proposed the "fair scheduling" that considers a relaxed version of the Liu's problem in which tasks are not restricted to using zero or one resource at a time, i.e. a resource may be shared by several tasks. But this approach cannot be applied to our problem because, contrary to Baruah's problem, our resources are processors that cannot be shared. Xu [3] and Shirazi [4] consider independent task sets where individual tasks are further divided into subtasks with precedence relation. Leung [5] provides a lot of results to solve the multiprocessor periodic scheduling with precedence relation but he considers that all the tasks have the same period. More recently, Baruah [6] considers the non-preemptive scheduling of periodic tasks onto multiprocessor problem using the EDF scheduling algorithm, but there are no precedences between tasks. We remind that, the applications we are interested in require both precedence and periodicity constraints.

Generally this kind of problem is solved with partitioning methods. Partitioning problems are analogous to the bin-packing problem known NP-hard [7]. Peng and Shin [8] present an optimal branch-and-bound algorithm but it is extremely slow, and thus, not usable for realistic applications, whereas heuristics approaches are more rapid despite their sub-optimal results. In this paper we deal with non-preemptive scheduling of tasks onto multiprocessor with both precedence relation and periodicity constraints taking into account the inter-processor communication execution times. There exists a lot of heuristics using back-tracking which solve problems close to ours. For example [9] gives such a heuristic based on "Simulated Anealing" technique, similarly [10] gives a heuristic based on "Genetic Algorithm" technique. Unfortunately, these kind of heuristics although they are more rapid than optimal algorithms, remind less rapid than greedy heuristics.

The main objective of our researches is to propose rapid heuristics because the realistic applications our industrial partners deal with, are very complex. These applications are usually composed of several thousands of tasks and several tens of processors. This complexity leads to choose first greedy heuristics to obtain results in a realistic time. Of course, these results may be used as an initial solution for other less rapid heuristics using back-tracking. In this paper

we focus on the former type of heuristic. Rapid heuristics which provides bad quality results actually do not have interest. This is why another objective is to propose a heuristic which produces effective results. The proposed heuristic will tend on the one hand to find a schedule which satisfies all the constraints, and on the other hand which minimizes the global execution time of the system. It is an extension of the heuristic proposed in [11] greatly improving the scheduling success ratio which is the ratio between the successfully scheduled systems and all the tested systems. Also, we mathematically prove the principles of the assignment algorithm through a probabilistic study. It was not relevent to compare our heuristic with related heuristics cited before because they do not deal with exactly the same problem, however we compared it with an optimal "Branch and Cut" algorithm which represents the most suitable optimal approach for this kind of problems.

The rest of the paper is organized as follows: the next section describes the model with precedence and periodicity constraints that we use. Then, we present some important issues relatively to periodicities, and a schedulability study. Section 3 presents the proposed heuristic. In section 4, a study of complexity is performed. A performance evaluation is discussed in section 5. Finally, Section 6 presents the conclusion of the paper and directions for future work.

2 Model and Schedulability

2.1 Model

We deal with systems of real-time tasks with precedence and strict periodicity constraints. A task τ is characterized by a period T_τ, a worst case execution time (WCET) C_τ with $C_\tau \leq T_\tau$, and a start time s_τ. Strict periodicity constraint means that if the periodic task τ has period T then $\forall i \in N, (s_{\tau_{i+1}} - s_{\tau_i}) = T$, where τ_i and τ_{i+1} are the i^{th} and the $(i+1)^{th}$ repetitions of the task τ, and s_{τ_i} and $s_{\tau_{i+1}}$ are their start times.

The precedences between tasks are represented by a directed acyclic graph (DAG) noticed G which is the pair (V, E). V is the set of tasks τ, characterized as above, and $E \subseteq V \times V$ the set of edges which represents the precedence (dependence) constraints between tasks. Therefore, the directed pair of tasks $(\tau, \rho) \in E$ means that ρ must be scheduled, only if τ was already scheduled and we have $s_\tau + C_\tau \leq s_\rho$. The multiprocessor architecture is represented with a undirected graph (V, E) where V is the set of processors and communication media, $E \subseteq V \times V$ the set of edges. Each edge represents a connection (electric wires) between a processor and a communication medium.

Notice that, since to schedule synchronous dataflow (SDF) graphs onto multiprocessors, a DAG is constructed from the original SDF graph [12] our proposed heuristic would also be applicable.

2.2 Periodicity Issues

As shown in [13] to analyse a system composed of non-preemptive periodic tasks it is enough to study its behavior for a time interval equal to the least common multiple (LCM) of all the task periods, called the hyper-period. Consequently, each task of the initial system will be repeated according to the ratio between its period and the hyper-period. Notice that in general, the value of the hyper-period is not large due to the relatively small number of sensors and actuators which impose their periods to the tasks [14]. Thus, the resulting system we have to deal with will not be significantly larger than the initial one

As explained in [15] we also assume that the dependent tasks must be at the same period or at multiple periods in order for the consumer task to be able to receive the data sent by the producer task without some data being lost or duplicated. This restriction does not prevent the presence of tasks with non multiple periods in the same system, nevertheless these tasks must not be dependent.

2.3 Schedulability Study

When there is at least one schedule satisfying all constraints of the system, the system is said to be schedulable. Contrary to the multiprocessor scheduling problems without periodicity constraints, our problem does not have always a solution. In other words a system of tasks with precedence and periodicity constraints is either schedulable or not. The schedulability analysis of this kind of systems is very complex and it is very difficult to know if a system is schedulable or not before executing an optimal scheduling algorithm on this system. We talk about an optimal algorithm because for some complex system, they are the only algorithms which can find a scheduling whatever the time that they take to test all the possibilities. The difficulty is to find an assignment for each task to one of the processors in such way that this task can be repeated following its period, and all the repetitions are scheduled on the same processor. As we deal with rapid greedy heuristics, we need to develop an assignment strategy which is rapid and has a good success scheduling ratio at the same time. The analysis of periodic tasks systems being very hard, let us take two tasks among the tasks of a system. Theses two tasks are always schedulable onto two different processors (each task on one processor), but what happens if there is only one processor? These tasks are either schedulable onto this processor or not.

The condition for scheduling two tasks τ and ρ under strict periodicity constraints on the same processor is:

$$C_\tau + C_\rho \leq GCD(T_\tau, T_\rho) \quad (1)$$

GCD stands for Greatest Common Divisor.

There are three possible cases for these two tasks (all the variables are integrals):

1. their periods are co-prime: condition (1) becomes $C_\tau + C_\rho \leq 1$ and it is always false because $(C_\tau + C_\rho)$ is at least equal to (1+1=2) units of time. In this case the two tasks must not be assigned to two different processors;

2. their periods are the same or multiple: if $T_\tau \leq T_\rho$, (1) becomes $C_\tau + C_\rho \leq T_\tau$.

3. their periods are neither the same or multiple, nor co-prime, (1) must hold.

From both latter cases it is only necessary to consider case 2 since it is the more likelihood to be satisfied, and considering both decrease too much the speed of the assignment algorithm without increasing its effectiveness. We performed probabilistic studies which strengthen this idea.

Therefore, we propose to assign to the same processor the tasks of the same period or the tasks whose periods are multiple.

In addition, according to the fact that the dependent tasks must be at the same, or multiple periods, the latter proposition leads to reduce the global execution time of inter-processor communications.

3 Proposed Heuristic

We now describe in detail our heuristic. It is a greedy heuristic which sequences the three algorithms: assignment, unrolling, and scheduling. For each algorithm, a decision made at some step is never questioned during the following steps (no backtracking). The effectiveness of any greedy heuristic is based on the choice of the decisions. In our case the decisions are based on the issues discussed in the previous section, that is, the hyper-period, the tasks repetition, and the idea which tends to execute on the same processor the tasks of the same, or multiple, periods.

3.1 Assignment Algorithm

In a previous version of our heuristic [11], we made an increasing sort of tasks according to their periods, and we assigned all the tasks with the same period to the same processor. Then, each remaining task (not assigned) was assigned to the corresponding processor if its period is a multiple of the period of the last task assigned to this processor. As soon as one of these tasks cannot be assigned, the system was said not schedulable. Figure 1 depicts a simple example where the assignment is possible but this approach is not able to find it.

In order to cope with this situation, we propose here an assignment algorithm based on a mixed sort which takes into account both the increasing order and a priority level. The latter is given to every task relatively to the number of tasks the periods of which divide this task period. We determine for each task how many divisors has its period relatively to the periods of the other tasks of the system. This gives its priority level. Then, we perform an increasing sort according to the priority level, and when the priority levels are equal we perform an increasing sort according only to the period. It is worth noticing that one task may be assigned to several processors, but finally will be distributed and scheduled onto only one of them. Actually, this is achieved in the third algorithm of the proposed heuristic.

Figure 1-(B) gives an example of this mixed sort to compare with the first one given in figure 1-(A). The system is composed of four tasks with four different periods values: 2, 3, 6, 8, and an architecture graph with two processors connected with one communication medium. On figure 1-(A) the task periods are sorted by increasing order. We observe that the task with period 8 cannot be assigned to any processor because 8 is neither multiple of 6 nor of 3. On figure 1-(B) the periods are ordered by the mixed order. The priority level of periods 2 and 3 is 0 because these numbers are prime numbers. Period 8 has level 1 because it is a multiple of period 2, and period 6 has level 2 because it is multiple of period 2 and 3. Consequently, the system of tasks can be assigned to the two processors.

On a representative set of tasks graphs we observed that this improvement increases significantly the schedulability ratio of the proposed heuristic without increasing very much the complexity of the assignement algorithm, from $N(M+logN)$ to $N(N + M + logN)$ (N is the number of graph tasks and M is the number of processors in the architecture).

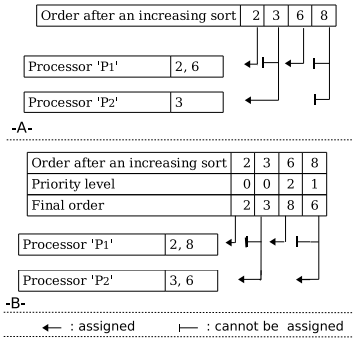

Figure 1: Tasks Assignment

Assignment, as we said, only takes into account period values (no execution times neither for computation nor for inter-processor communication). This means finding an assignment for the system does not ensure the schedulability of this system. In addition, when it is not possible to find an assignment, we consider that the system is not schedulable, and the heuristic stops here. The execution and communication times will be taking into account when the tasks are scheduled in the third algorithm.

3.2 Graph Unrolling Algorithm

This algorithm consists in repeating each task of the graph, n times. $n = (hp/T)$ where the period of this task is T and hp is the hyper-period [9]. In order to maintain the data transfers between tasks during the unrolling, we must add new edges between the repetitions of the same task, and between the repetitions of different tasks [11].

As we supposed in section 2 that data transfer can be carried out only between two tasks having the same or multiple periods, then adding the edges is performed as follows:

- a precedence edge is added between each two repetitions of the same task,
- between the repetitions of a producer task and a consumer task, as many edges as the repetitions of the producer task are added. If T_1 is the period of the producer task and T_2 is the period of the consumer task and $T_2 \geq T_1$, T_2/T_1 repetitions of the producer task are connected to each repetition of the consumer task. This is detailed with an example in [11].

3.3 Scheduling Algorithm

This algorithm distributes and schedules each task of the unrolled graph onto the processor where it has been assigned by the assignment algorithm. In case where the task was assigned to several processors the algorithm chooses the processor which minimizes the total execution time. At each main step of the scheduling algorithm, one task is selected from the set of schedulable tasks. This set contains tasks which have no predecessor in the graph of tasks, or which have all its predecessors already scheduled.

This algorithm on the one hand guarantees the periodicity constraints, and on the other hand minimizes the total execution time of the system. Indeed, to guarantee periodicity constraints means that to be scheduled onto a processor, a task must not prevent the scheduling of the repetitions of already scheduled tasks on this processor. Then, once a task has been scheduled, all its repetitions will have fixed start times according to the period of this task, and the repetition number of the task. For minimizing the total execution time, the algorithm uses the cost function proposed in [16] which takes into account the worst case execution times of each task and of the inter-processor communications due to the dependences between tasks, and its schedule flexibility. The cost function allows the total execution time of the system to be minimized. Algorithm 1 details the different steps of the scheduling algorithm.

When a task is scheduled onto a processor, if it depends from one or several tasks already scheduled on another processor, as much receive tasks as there are of dependences must be created and scheduled before this task which consumes the data. On the other hand a send task must be created and scheduled after the task which produces the data. The data transfer associated to the dependence is carried out by sending and receiving messages through the communication medium, which connects both processors where corresponding tasks are executed. The communication execution time specifies the time slipped by the start time of the sending task and the completion time of the receiving task. The data transfer mechanism is detailed in the next section.

Algorithm 1 Scheduling Algorithm

```
1:  Initialization of Ω the set of schedulable tasks
2:  while Ω is not empty do
3:      for i = 1 to schedulable tasks number do
4:          If the schedulable task satisfies the periodicity of the
            already scheduled tasks on one or several processors
            it has been assigned to, then compute the cost func-
            tion for the processors where the condition is satis-
            fied
5:      end for
6:      for i = 1 to schedulable tasks number do
7:          Take the minimal cost function already calculated
            for the schedulable task on all the processors to get
            the better one and to form ( schedulable task, best
            processor) pairs
8:      end for
9:      for i = 1 to the number of ( schedulable task, best
        processor) pairs do
10:         Choose the (schedulable task, best processor) pair
            which has the maximal cost function.
11:     end for
12:     Now, the schedulable task is definitely distributed on
        its best processor
13:     Schedule this schedulable task on its best processor,
        and schedule all messages tasks if needed
14:     Add to Ω the schedulable successors of this task (the
        tasks the predecessors of which have been already
        scheduled, or tasks without any other predecessor)
15:     Remove the scheduled task from Ω
16: end while
```

3.4 Data Transfer Mechanism

When two tasks are dependent, and have not the same period there are two possibilities. If the period of the consumer task is equal to n times the period of the producer task then the producer task must be executed n times compared to the consumer task, and the consumer task cannot start its execution until it has received all data from the n executions of the producer task (we have to precise that the produced data differ from one execution of the producer task to another execution therefore data are not duplicated). Reciprocally, if the period of the producer task is equal to n times the period of the con-

sumer task then the consumer task must be executed n times compared to the producer task. The unrolling algorithm exploits this data transfer mechanism.

3.5 Heuristic Implementation

The proposed heuristic has been implemented in SynDEx [17] a system level CAD software, for rapid prototyping and optimizing the implementation of distributed real-time embedded applications. This software have been used to implement the "manual driving" application for the Cycab (an intelligent automatic vehicle www-lara.inria.fr/cycaba) which involves two different periods.

4 Heuristic Complexity

If M is the number of processors in the architecture graph, and N is the number of tasks in the graph of tasks, the number of unrolled graph tasks is equal to $N_{unr} = \sum_{i=1}^{N} hp/T(\tau_i)$ with τ_i a task, $T(\tau_i)$ its period, and hp the hyper-period. The complexity of the proposed heuristic is $N_{unr}^2 M$. The complexity explodes as soon as N_{unr} becomes large. However, as we said in section 2, the number of different task periods among tasks is relatively small which reduces considerably N_{unr} value.

5 Performance Analysis

In order to analyse its performance the proposed heuristic was compared to an optimal "Branch and Cut" (B&C) algorithm that we had already implemented.

5.1 Optimal Branch and Cut Algorithm

The B&C algorithm [18] solves the multiprocessor scheduling problem described above and is able to find a schedule if it exists whatever the number of different periods of tasks in the system. At each step, our B&C algorithm tests for each task if it is possible to distribute it onto a processor where other tasks have been already distributed then scheduled. If this condition is satisfied this task is distributed and scheduled on this processor otherwise the algorithm back-tracks.

5.2 Results

When we generated multiperiodic systems for performance analysis we took into consideration the periodicity issues presented in section 2.2. Since there is no rule to determine the minimum number of processors for the system being schedulable, we executed the heuristic and the optimal algorithm with an initial number of processors, and increased it until the system became schedulable. Thus, we generated automatically tasks graphs taking into account the periodic issues with dependent and non dependent tasks. The content itself of the task has no impact on the scheduling, only its WCET and its period are relevant. Also, we generated systems such that the number of different periods is not large relatively to the number of tasks, however we generated systems with all the possible cases (multiple and not multiple period) in order to obtain more realistic results. In addition, the architecture graphs were generated according to a star topology meaning that any two processors can communicate through the medium without using intermediate processors (no routing).

There are a lot of parameters which may affect the performance of distributed real-time systems when comparing the heuristic with the optimal algorithm. Consequently, we performed two kinds of tests. The first one consisted in comparing the scheduling success ratio for the heuristic and the optimal algorithm on different systems. This test has been performed as follows: we compute for each system the value of λ which is the ratio between the number of processors and the number of different and non multiple periods. Then, we gather all the systems with the same λ, and we execute for them the optimal algorithm and the proposed heuristic. Finally, we compute the scheduling success ratio for the systems of each λ by using the results of the previous step. This method allows us to underline the impact of the architecture in terms of number of processors, and of the number of periods of tasks. The diagram of figure 2 depicts the evolution of the success ratio according to the variation of λ. It shows that our heuristic has in average 87% of scheduling success ratio for all the λ values, but when $\lambda \geq 0.5$ this ratio becomes 94.5 %. It means that our heuristic is more effective when the number of different and non multiple periods is less or equal to the number of processors. As it is explained in section 2.2 this case is the common case in the industry.

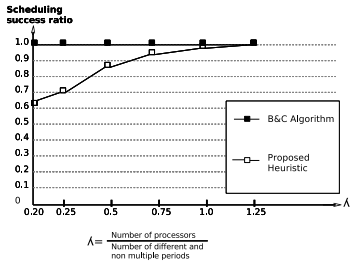

Figure 2: Scheduling Success Ratio Comparison

In the second test, the speed of the heuristic and of the optimal algorithm are evaluated by varying the size of the systems (both number graph tasks and number of processors). The diagram of figure 3 shows that the optimal algorithm explodes very quickly whereas our heuristic keeps a reasonable execution time. Notice that the algorithm execution time follows a logarithmic scale.

These results show that our heuristic is effective in terms of

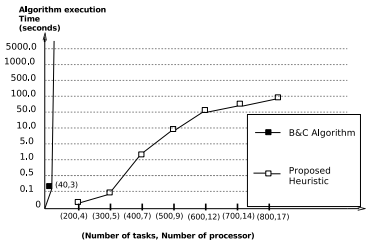

Figure 3: Speed Comparison

scheduling success ratio and speed.

6 Conclusion and Further Research

We presented an effective greedy heuristic in terms of scheduling success ratio and speed, to solve the non-preemptive multiprocessor scheduling problem with precedence and strict periodicity constraints.

The heuristic is composed of three algorithms. The first one which results from a schedulability study presented in this paper assigns the graph tasks to the processors according to a mixed sort of their periods. This new sort allowed us to better deal with the different periods of the tasks. The second algorithm handles the periodicity of the tasks by repeating them and adding the missing edges to the graph. The third algorithm distributes and schedules the tasks by using a cost function which leads to minimize the total execution time of the system. Two kinds of tests have been performed in order to compare our heuristic to the optimal Branch and Cut algorithm. The results showed the effectiveness of our heuristic.

We plan to investigate other distributions and scheduling heuristics, and especially we plan to study meta-heuristics which will certainly have a best scheduling success ratio but will increase the speed.

References

[1] C. L. Liu. Scheduling algorithms for multiprocessors in a hard real-time environment. In *JPL Space Programs Summary 37-60*, volume 2, pages 28–37, November 1969.

[2] S. K. Baruah. Fairness in periodic real-time scheduling. In *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS '95)*, page 200, Washington, DC, USA, 1995. IEEE Computer Society.

[3] J. Xu. Multiprocessor scheduling of processes with release times, deadlines, precedence, and exclusion relations. *IEEE Transactions, on Parallel Distibuted Systems*, 19(2), February 1993.

[4] S. Ronngren and A. Shirazi. Static multiprocessor scheduling of periodic real-time tasks with precedence constraints and communication costs. In *Proceeding of the 28th Annual Hawaii International conference on system Sciences*, page 143, January 1995.

[5] J. Leung. *Handbook of Scheduling*. Published by CRC Press, Boca Raton, FL, USA, May 2004.

[6] Sanjoy K. Baruah. The non-preemptive scheduling of periodic tasks upon multiprocessors. *Real-Time Systems*, 32(1-2):9–20, 2006.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability. A guide to the theory of NP-completeness*. W. H. Freeman, 1979.

[8] D.-T. Peng and K. G. Shin. Optimal scheduling of cooperative tasks in a distributed system using an enumerative method. *IEEE Trans. Softw. Eng.*, 19(3):253–267, 1993.

[9] K. Ramamritham. Allocation and scheduling of precedence-related periodic tasks. *IEEE Transactions on Parallel and Distributed Systems*, 6(4), April 1995.

[10] S. Wu, H. Yu, S. Jin, K. Lin, and G. Schiavone. An incremental genetic algorithm approach to multiprocessor scheduling. *IEEE Transactions on Parallel Distibuted Systems*, 15(9), September 2004.

[11] O. Kermia, L. Cucu, and Y. Sorel. Non-preemptive multiprocessor static scheduling for systems with precedence and strict periodicity constraints. In *Proceedings of the 10th International Workshop On Project Management and Scheduling, PMS'06*, Posnan, Poland, April 2006.

[12] J. Pino and E. Lee. Hierarchical static scheduling of dataflow graphs onto multiple processors. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, Detroit, Michigan*, 1995.

[13] K. Danne and M. Platzner. A heuristic approach to schedule periodic real-time tasks on reconfigurable hardware. In *Proceedings of the International Conference on Field Programmable Logic and Applications*, pages 568–573, August 2005.

[14] M. Alfano, A. Di-Stefano, L. Lo-Bello, O. Mirabella, and J.H. Stewman. An expert system for planning real-time distributed task allocation. In *Proceedings of the Florida AI Research Symposium*, Key West, FL, USA, May 1996.

[15] T. F. Abdelzaher and K. G. Shin. Period-based load partitioning and assignment for large real-time applications. *IEEE Transactions on Computers*, 49 (1):81–87, January 2000.

[16] T. Grandpierre, C. Lavarenne, and Y. Sorel. Optimized rapid prototyping for real-time embedded heterogeneous multiprocessors. In *Proceedings of 7th International Workshop on Hardware/Software Co-Design*, Rome, Italy, May 1999.

[17] Y. Sorel. Syndex: System-level cad software for optimizing distributed real-time embedded systems. *Journal ERCIM News*, 59:68–69, October 2004.

[18] John E. Mitchell. Branch-and-cut algorithms for combinatorial optimization problems. In *Handbook of Applied Optimization*, pages 65–77. Oxford University Press, January 2002.