Operations Research Letters 45 (2017) 652–658

Contents lists available at ScienceDirect

Operations Research Letters

journal homepage: www.elsevier.com/locate/orl

Generation of feasible integer solutions on a massively parallel computer using the feasibility pump

Utku Koc [a,b,*], Sanjay Mehrotra [a]

[a] Northwestern University, Evanston, IL, USA
[b] MEF University, Istanbul, Turkey

ARTICLE INFO

Article history:
Received 19 July 2016
Received in revised form 3 October 2017
Accepted 3 October 2017
Available online 19 October 2017

Keywords:
Mixed integer programming
Parallel optimization
Feasibility pump

ABSTRACT

We present an approach to parallelize generation of feasible mixed integer solutions of mixed integer linear programs in distributed memory high performance computing environments. This approach combines a parallel framework with feasibility pump (FP) as the rounding heuristic. It runs multiple FP instances with different starting solutions concurrently, while allowing them to share information. Our computational results suggest that the improvement resulting from parallelization using our approach is statistically significant.

1. Introduction

In this study we consider the problem of generating high quality feasible solutions for unstructured Mixed Integer Linear Programs (MILPs) in a parallel computational environment. We refer the interested reader to Lodi [18] for a recent review of MILP literature. Generating high quality feasible solutions quickly is important in practice. Additionally, availability of feasible solutions with close to optimal objective value, may help reduce the number of nodes in the branch and bound (B&B) tree in a branch and cut algorithm. We propose a scheme that can use multiple heuristics with various parameter settings in parallel. Specifically, we empirically investigate the use of Feasibility Pump (FP) to find feasible solutions for unstructured MILPs in a parallel framework.

The motivation of this study is the emerging computing environment. The clock speed of the high-tech processors is more or less stable for the past few years. Computer technology is now mainly focused on increasing the number of processors and memory. With this in mind, we move to a new era of developing parallel algorithms for a variety of problems for desktop and high performance computing (HPC). From a practical point of view, it is important to solve a problem or identify a good solution within a reasonable amount of wall-clock time, de-emphasizing the CPU-time used.

For MILPs, a way to use the power of parallel computing is to search the branch and bound tree in parallel. Koch et al. [17] discuss that the speed up of a B&B algorithm is around 20,000 compared to a sequential run, even if a million cores are used to search the B&B tree. They discuss that the dis-proportionality in the performance is mainly due to the communication overhead, performance effect of the redundant work, and idle time due to latency/contention/starvation.

The FP algorithm is a constructive heuristic for MILPs that starts from a feasible solution x of the continuous relaxation, searches for another solution $\hat{x}$ that is as close as possible to a rounded solution of x (which is infeasible but integral) by solving an ℓ_1 norm minimization problem. The algorithm continues until a feasible integral solution is found. The FP heuristic was first proposed by Fischetti et al. [10] for 0–1 MILPs. An extension to general MILPs is proposed by Bertacco et al. [7]. By a modification of the objective function, Achterberg and Berthold [1] found better feasible solutions (Objective FP). Fischetti and Salvagnin proposed a different rounding heuristic by using constraint propagation techniques after rounding some of the variables [13]. Baena and Castro [3] extended the FP, so that the integer point is obtained by rounding a point on the (feasible) line segment between the computed feasible point and the analytic center for the relaxed LP. Huang and Mehrotra studied a combination of different types of random walks and FP in which the FP algorithm is used as the rounding procedure for interior random points. They generate feasible solutions for MILPs [14] and Mixed Integer Convex Programs (MICPs) [15]. Fischetti et al. studied to increase the stability of the root node LP solutions through parallel independent runs with diverse initial conditions [12]. Munguia et al. [19] combine the use of parallelization and simple large neighborhood search schemes to generate feasible solutions for MILPs.

* Corresponding author at: MEF University, Istanbul, Turkey.
E-mail addresses: utku.koc@mef.edu.tr (U. Koc), mehrotra@northwestern.edu (S. Mehrotra).

https://doi.org/10.1016/j.orl.2017.10.003
0167-6377/© 2017 Elsevier B.V. All rights reserved.

There are several other heuristics for finding feasible solutions for MILP problems that can be used as a part of a parallel implementation. Among them, Pivot-and-Complement [5,6] performs simplex like pivots to get slack variable into the basis and integer variables out of a basis. Another heuristic for 0–1 MILP is OCTANE, which uses enumeration techniques on extended facets of the octahedron [4]. Fischetti and Lodi propose a local search algorithm [11] to improve an incumbent solution. Relaxation Induced Neighborhood Search (RINS) heuristic solves sufficiently smaller sub-MILPs to improve an incumbent solution [9], and the use of random-walks was investigated in [14,15].

In this study, we provide a parallel framework in which multiple feasibility heuristics starting from different solutions can communicate and share information. We assess the value of parallelization independent of the increase in the CPU-time and provide a parallel framework that can use multiple parameters for feasibility heuristics. Each parallel subroutine uses a different rounding scheme so that the most fractional variables are rounded in an enumerative fashion independently. This study is the first of its kind in terms of using many cores to generate feasible integer solutions in parallel in a distributed memory environment with many cores.

The rest of the paper is organized as follows: we describe our parallel heuristic framework in Section 2. Details of the rounding procedure are given in Section 3. Section 4 gives the implementation details of the proposed algorithms. The computational results are given in Section 5. We show that the proposed approach is effective up to 128 cores in today's HPC environment. Finally, we conclude in Section 6.

2. A concurrent framework for finding feasible solutions for MILPs

In our approach, we run multiple feasibility heuristics in parallel. We refer to the algorithms running in different processors as subroutines. Each parallel subroutine uses a different random number seed with different starting solutions. One may also run different feasibility heuristics in parallel. Moreover, any combination of parameter settings, rounding methods, and anti-cycling rules are also valid. Note that, even if all the subroutines start from the same solution and run independently, final integer solutions may still be different. This is because multiple instances can take different paths in the course of the parallel subroutines. Whenever one of the subroutines finds a feasible solution, it broadcasts the objective function value to others via the master. Then, all subroutines continue their search with a new and better objective cut-off constraint. Thus, the information gained in one of the subroutines is shared with the rest to enhance their search. This is an important feature of our concurrent optimization approach. All subroutines update themselves as soon as the first feasible solution is found. In this study, as a proof of concept, we use FP as the rounding procedure of the subroutines of our concurrent feasibility heuristic.

Regarding the communication during the run time, one may use the so called master/slave topology. In this paradigm, the master controls the overall course of the algorithm. Slave programs, on the other hand, follow the commands from the master, run the instances of the heuristic, and return integer solution(s) to the master, if any. The role of the master is distributing inputs to and collecting results from the slaves. The main algorithm that runs at the master is presented in Algorithm 2.1.

We illustrate the algorithm running at the slaves in Algorithm 2.2. Each slave uses a different random number seed and may run a different variant of a heuristic. At each iteration of Algorithm 2.2 , the slave subroutine receives relevant information from the master (if any), updates itself with the new information, creates a starting solution for the algorithms depending on the type of heuristic it is running and sends the integer solutions to the master, if any.

Algorithm 2.1 *Parallel Feasibility-Pump Running in Master*

```
Input: a MILP $\min\{c^T x : Ax \geq b, x \in \mathbb{R}^n, x_j \text{ integer } \forall j \in I\}$, number
    of slaves each heuristic will run
Output: an integer solution to the above MILP
 1: Spawn Slaves
 2: Set $LB = \min\{c^T x : Ax \geq b, x \in \mathbb{R}^n\}$ and $UB = \infty$
 3: Inform slaves about heuristic to run and $LB$
 4: while termination criteria not met do
 5:    Collect results
 6:    if One of the slaves returns an integer solution then
 7:       Update $UB$ = minimum of the slaves
 8:       Inform slaves about the new $UB$
 9:    end if
10: end while
11: Exit all the slaves and return best integer so far
```

Algorithm 2.2 *Parallel Heuristic Subroutine Running in Slaves*

```
Input: a MILP, $UB$
Output: an integer solution to the MILP
 1: Receive the type of the heuristic to run and LB form the Master
 2: while not exited by the master do
 3:    Listen to the master for information ($UB$)
 4:    Update $RHS$ of the objective cut-off constraint
 5:    Update with respect to the heuristic variant
 6:    Create a starting solution $x$
 7:    Run heuristic starting from $x$
 8:    Broadcast best integer solution
 9: end while
```

The heuristic subroutine continues until predetermined criteria are met, or the master provides new information.

The variants of the heuristic subroutines differ in Steps 5–7 of Algorithm 2.2. The update procedure, generations of starting solutions, and running conditions of the heuristics depend on the heuristic itself and information provided by the master.

3. Variants of FP heuristic

In this section we describe the details of the rounding subroutine, as well as the generation of the starting solutions for rounding. We start with the details of the basic FP algorithm as the rounding procedure.

3.1. Basic and objective FP algorithms

The FP algorithm starts from a solution x, searches for another solution $\hat{x}$ that is as close as possible to a rounded solution of x ($\tilde{x}$) by solving an ℓ_1 norm minimization problem of the form:

$$\min\left\{\Delta(x,\tilde{x}) = \sum_{j\in I}|x_j - \tilde{x}_j| : Ax \geq b, c^T x \leq RHS, x \in \mathbb{R}^n\right\}$$

where $\Delta(x,\tilde{x})$ is the ℓ_1 norm distance, $Ax \geq b$ is the constraint set defined by the original MILP and $c^T x \leq RHS$ is the objective cut-off constraint. For problems with general integer variables, Δ is defined by adding artificial variables. At each attempt to solve the problem, ℓ_1 norm distance function Δ is updated with respect to the rounded solution $\tilde{x}$. In this heuristic, one needs to decide on how the starting solutions (x^k) and rounding procedure ($\tilde{x}$) are defined at each iteration k.

Using a normalized convex combination of the original objective function and the above ℓ_1 norm objective, one can generate better quality solutions (Objective-FP) [1]. The idea is to focus more