Improving Wedelin's Heuristic with Sensitivity Analysis

Sara Maqrot, Simon de Givry, Gauthier Quesnel, Marc Tchamitchian

Improving Wedelin's Heuristic with Sensitivity Analysis

Sara Maqrot[1], Simon de Givry[1], Marc Tchamitchian[2], Gauthier Quesnel[1]

[1] UR 875 MIAT, Université de Toulouse, INRA, Castanet-Tolosan, France

[2] UR 767 Ecodéveloppement, INRA, Avignon, France

{firstname.lastname}@inra.fr

Keywords : *integer programming, heuristics, Lagrangian relaxation, sensitivity analysis*

Abstract

Heuristic is one of the important techniques designed to find quickly good feasible solutions for hard integer programs. Most heuristics depend on a solution of the relaxed linear program. Another approach, Lagrangian relaxation offers a number of important advantages over linear programming [4], namely it is extremely fast for solving large problems. One of the Lagrangian based heuristics is Wedelin's heuristic [10], which works for the limited class of 0-1 integer programs. It is designed for problems with a specific form ($min_{x \in \{0,1\}^n}\{c^T x | Ax = b\}$ with coefficients of A in $\{0, 1\}$ and b integral) which prevents its applicability on many problems. Thus to tackle problems with more general structures, [1] presents a generalized Wedelin's heuristic for integer programming. The performance of this method depends crucially on the choice of its numerous parameters (number of iterations, degree of approximation, history of the *preference matrix*, and others). To adjust these parameters and learn which ones have important influence on whether a solution is found and its quality, we conduct sensitivity analysis combined with a metaheuristic. We choose the Morris method [8] to select parameters providing a feasible/good solutions on a family of instances, and the *genetic optimization using derivatives* (genoud) method [7] to find the best solution in limited time for a specific instance. The Morris method consists in discretizing the input space for each parameter, then performing a given number of *One At a Time* random designs (each input parameter is varied while fixing the others). The repetition of these steps allows the estimation of elementary effects for each parameter, and consequently the sensitivity indices [5]. genoud combines an evolutionary algorithm method with a derivative based (quasi-Newton) method to solve difficult optimization problems. These difficulties often arise when the objective function is a nonlinear function of the continuous parameters and not globally concave having multiple local optima [7].

We have implemented a C++ parallel version of Wedelin's heuristic based on [1]. The solver called baryonyx[1] has two execution modes: solver mode and optimizer mode. In the solver mode, baryonyx runs once trying to satisfy all the constraints. In the optimizer mode, it runs in parallel according to the number of processors, and tries to satisfy all the constraints and to optimize the solution at each run, reporting the best solution found for all runs when it reaches its time limit. Concerning parameters, Morris and genoud methods are implemented in R packages. We use Morris package to find useful parameters. Once found, we fix other parameters and we let genoud adjusts the useful ones in order to get the best solution within a given time limit. We compare baryonyx with exact solver IBM ILOG cplex and with two local search methods: a 4-flip neighborhood local search algorithm [9] and an hybrid mathematical programming solver LocalSolver [2] which is a simulated annealing based on ejection chain moves specialized for maintaining the feasibility of Boolean constraints and an efficient incremental evaluation using a directed acyclic graph. The following Tables show that baryonyx is competitive with the existing solvers on a Set Partitioning Problem (SPP) benchmark [3], but has difficulties on a Mixed Fruit-Vegetable Crop Allocation Problem (MFVCAP) [6].

[1] https://github.com/quesnel/baryonyx

(600/3600 sec CPU time limit on 2.5GHz Intel XEON using 1 core, except 7200s Borndörfer)

SPP Instance	Cplex12.6	LocalSolver3.1	Borndörfer	Umetani	baryonyx
v0415 (600s)	**2,429,415**	**2,429,415**	2,429,415	2,429,568	2,432,717
v0416 (600s)	**2,725,602**	2,728,391	**2,725,602**	2,726,156	2,730,390
v0417 (600s)	**2,611,518**	2,617,387	**2,611,518**	**2,611,518**	2,614,359
v0418 (600s)	**2,845,425**	2,846,265	**2,845,425**	**2,845,425**	2,848,692
v0419 (600s)	**2,590,326**	2,590,511	**2,590,326**	**2,590,326**	2,592,139
v0420 (600s)	**1,696,889**	**1,696,889**	**1,696,889**	**1,696,889**	1,697,954
v0421 (600s)	**1,853,951**	**1,853,951**	**1,853,951**	**1,853,951**	1,855,344
v1616 (600s)	**1,006,460**	1,051,749	**1,006,460**	1,007,402	1,019,799
v1617 (600s)	**1,102,586**	1,181,503	**1,102,586**	1,103,651	1,120,193
v1618 (600s)	**1,153,871**	1,221,162	1,154,458	1,155,986	1,172,519
v1619 (600s)	**1,156,338**	1,221,960	**1,156,338**	1,157,537	∞
v1620 (600s)	**1,140,604**	1,230,809	**1,140,604**	1,141,976	1,155,197
v1621 (600s)	**825,563**	838,192	**825,563**	825,605	832,545
v1622 (600s)	**793,445**	805,346	**793,445**	793,708	801,029
t0415 (600s)	**5,339,422**	∞	5,590,096	5,572,626	5,404,140
t0416 (600s)	6,093,843	∞	6,130,271	**6,088,264**	6,093,843
t0417 (600s)	**5,951,357**	∞	6,043,157	6,024,760	5,953,029
t0418 (600s)	6,550,898	∞	6,550,898	**6,446,019**	6,447,571
t0419 (600s)	**5,907,874**	∞	5,916,956	5,910,913	5,910,913
t0420 (600s)	4,276,444	∞	4,276,444	∞	**4,155,076**
t0421 (600s)	**4,290,809**	∞	4,354,411	**4,290,809**	4,313,091
t1716 (3600s)	184,160	∞	161,636	165,972	**157,442**
t1717 (3600s)	200,300	∞	184,692	180,757	**167,063**
t1718 (3600s)	183,349	∞	**162,992**	174338	172,652
t1719 (3600s)	203,839	∞	187,677	184,354	**179,400**
t1720 (3600s)	179,283	∞	172,752	181,868	**163,432**
t1721 (3600s)	136,092	202,520	127,424	130,047	**123,626**
t1722 (3600s)	120,303	∞	122,472	**114,508**	118,242

(3600 sec CPU time limit on 2.5GHz Intel XEON using 10 cores)

MFVCAP Instance	Cplex12.7 MIP/Benders		Cplex12.7 BQP		LocalSolver7.5	baryonyx
	first solution	best sol.	first sol.	best sol.		
Equilibrate 50 × 50	316,500	**112,490**	587,710	349,050	118,850	329,290

References

[1] Olivier Bastert, Benjamin Hummel, and Sven de Vries. A generalized Wedelin heuristic for integer programming. *INFORMS Journal on Computing*, 22(1):93–107, 2010.

[2] T Benoist, B Estellon, F Gardi, R Megel, and K Nouioua. Localsolver 1. x: a black-box local-search solver for 0-1 programming. *4OR*, 9(3):299–316, 2011.

[3] R. Borndörfer, M. Grötschel, F. Klostermeier, and C. Küttner. *Telebus Berlin: Vehicle Scheduling in a Dial-a-Ride System.* Springer Berlin Heidelberg, 1999.

[4] Marshall L. Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management science*, 50(12_supplement):1861–1871, 2004.

[5] Bertrand Iooss and Paul Lemaître. A review on global sensitivity analysis methods. In *Uncertainty Manag. in Simul.-Optim. of Complex Systems*, pages 101–122. Springer, 2015.

[6] S Maqrot, S de Givry, G Quesnel, and M Tchamitchian. A Mixed Integer Programming Reformulation of the Mixed Fruit-Vegetable Crop Allocation Problem. In *Advances in Artificial Intelligence: From Theory to Practice*, pages 237–250, Arras, France, 2017.

[7] Walter R. Mebane Jr and Jasjeet S. Sekhon. Genetic optimization using derivatives: the rgenoud package for R. *Journal of Statistical Software*, 42(11):1–26, 2011.

[8] Max D. Morris. Factorial sampling plans for preliminary computational experiments. *Technometrics*, 33(2):161–174, 1991.

[9] Shunji Umetani. Exploiting variable associations to configure efficient local search algorithms in large-scale binary integer programs. *EJOR*, 263(1):72–81, 2017.

[10] Dag Wedelin. An algorithm for large scale 0/1 integer programming with application to airline crew scheduling. *Annals of operations research*, 57(1):283–301, 1995.