Combining Model-Based EAs for Mixed-Integer Problems

Krzysztof L. Sadowski[1], Dirk Thierens[1], and Peter A.N. Bosman[2]

[1] Utrecht University, The Netherlands
[2] CWI Amsterdam, The Netherlands

Abstract. A key characteristic of Mixed-Integer (MI) problems is the presence of both continuous and discrete problem variables. These variables can interact in various ways, resulting in challenging optimization problems. In this paper, we study the design of an algorithm that combines the strengths of LTGA and iAMaLGaM: state-of-the-art model-building EAs designed for discrete and continuous search spaces, respectively. We examine and discuss issues which emerge when trying to integrate those two algorithms into the MI setting. Our considerations lead to a design of a new algorithm for solving MI problems, which we motivate and compare with alternative approaches.

1 Introduction

Mixed-Integer (MI) optimization problems arise in many real-world application domains. A key characteristic of MI problems is the presence of both continuous and discrete problem variables. Many studies exist on dealing with either continuous or discrete search spaces only. We are interested in studying if and how approaches originally designed for real or discrete domains only can be integrated for the mixed-integer landscapes.

More specifically, we consider two state-of-the-art model building EAs: LTGA [7] and iAMaLGaM [1]. Both were previously shown to exhibit excellent polynomial scale-up behavior on various well-known black-box benchmark problems. We wish to study if making use of the model building and learning abilities of both these algorithms can be applied to MI problems while retaining some of the excellent scale-up behavior. The model-building nature of these algorithms allows us to consider black-box problems where no prior information about a problem structure is known. Some research on solving MI problems with EDAs has been discussed in [4], but was limited in terms of possible variable dependencies.

We introduce an integrated implementation which relies on interleaving the model-building capabilities of both EAs. Integrative dependency processing is the holy grail of an approach using model-building algorithms to solve MI problems. It is important to first understand the capacity and limitations of an approach that interleaves existing individual models, as it allows us to better understand the requirements for integrative dependency processing. A crucial aspect of designing such algorithm is determining a way of maintaining a proper balance between structure learning and offspring creation done by the two independent models. Obtaining such balance is a difficult task, as many

T. Bartz-Beielstein et al. (Eds.): PPSN XIII 2014, LNCS 8672, pp. 342–351, 2014.

challenges arise when dealing with MI landscapes, which do not exist in only discrete or continuous spaces. Because of the nature of MI landscapes, performing independent learning with different models for the continuous and discrete variables can easily lead to premature convergence when some variables are not sufficiently explored, or to over-exploration when too much focus is given to some variables. How difficult is it to achieve a proper evaluation balance and adequate scalability as the problem size increases? Is it even possible to solve dependent problems where continuous variables interact with the discrete ones, while using integrated but independently learning models?

In order to answer these and other questions, we design and study mixed-integer landscapes with different levels of variable interactions: no interactions, binary and/or continuous interactions only, and interactions between both types of variables. We identify various problematic issues, and design methodologies to counteract those issues. We also examine the performance of the existing Mixed-Integer Evolution Strategy (MIES) [3] [5] on our benchmark set.

2 Background

Our approach to solving Mixed-Integer problems focuses on bringing together two model-building algorithms, LTGA and iAMaLGaM, and carefully integrating them.

2.1 LTGA

The Linkage Tree Genetic Algorithm (LTGA) is a state-of-the-art model building GA designed for solving discrete problems [7] [8]. It makes use of a hierarchical clustering algorithm in each generation in order to learn variable dependencies, which are represented via a linkage tree. In this model, each node of the linkage tree is a subset between one and $l_d - 1$ problem variables which form an important building block for the solution.

During each generation, LTGA iterates over all solutions in the population in an attempt to improve them: For each solution the linkage tree is traversed and each subset of the tree is used as a crossover mask between a donor and the parent solution. A donor is selected randomly from the population for each subset mask. In other words, the values of variables clustered together at a given node of the linkage tree are copied from a donor onto the parent solution. A result of each such crossover is immediately evaluated. If the resulting offspring solution is better or equal than its parent, it instantly replaces the parent. Otherwise, the offspring is discarded. This process is repeated until all the linkage tree nodes are processed. This algorithm has been shown to work very efficiently for various discrete problems [8].

2.2 iAMaLGaM

Incremental Adapted Maximum-Likelihood Gaussian Model Iterated Density Estimation Evolutionary Algorithm (iAMaLGaM) is a state-of-the-art EDA for real-valued black-box optimization (BBO) [1]. Following the general EDA paradigm, iAMaLGaM estimates a probability distribution every generation from the selected solutions and

generates new solutions by sampling the estimated distribution. The probability distribution used in iAMaLGaM is the Gaussian distribution. The mean vector and covariance matrix are estimated incrementally using memory decay on maximum-likelihood estimates. Risk of premature convergence is counteracted by a mechanism which scales up the co-variance matrix when needed. Finally, Anticipated Mean Shift procedure is implemented to improve the algorithm behavior in slope-like regions of the search space. All these factors contribute to iAMaLGaM achieving very good scale-up and rotation-invariant behavior on many well known BBO benchmarks [2].

3 Integrated Algorithm

A solution to a mixed problem with a total problem length $l = l_d + l_c$, where l_d and l_c are the number of discrete and continuous variables respectively, is of the form:

$$X = X_d X_c = d_0 ... d_{l_d - 1}\ c_0 ... c_{l_c - 1}$$

where $d_i \in \{0, 1\}, c_i \in \mathbb{R}$ and X_d, X_c are the sets of all discrete and continuous variables, respectively.

Our integrated algorithm builds models over the two subspaces independently. Models can be build and exploited in various ways. A balance between the rates at which this happens is likely to play an important role in the convergence properties of the algorithm. The models in question have some common properties, which we can use as a backbone for integration. Both models attempt to improve and generate new offspring for each solution in the population during one generation, and learn a new model at the beginning of a next generation. However, after a new model is generated, iAMaLGaM creates the new continuous solutions by sampling from the new model. Once all the offspring solutions are created, the generation ends. This means that for a population of size n, iAMaLGaM performs n function evaluations within one generation.This differs from the generational procedure of LTGA. The linkage tree contains $2l_d - 1$ nodes. Following the variation procedure of LTGA, a solution may need to be evaluated $2l_d - 1$ times, giving the upper bound of $n * (2l_d - 1)$ evaluations overall during one entire generation. A straightforward approach of directly merging these models by keeping their generations synchronized might not work well, as depending on the ratio of discrete to continuous variables in a MI setting, one model could dominate the other by exploring some areas of the problem too heavily, while leaving other regions potentially not explored enough. This could lead to premature convergence or unnecessary over-exploration of the search space. To address this potential imbalance, we introduce the integrated EA in Figure 1. In this algorithm, the generational progress of the different models is not the same, and takes into account the proportions between the number of discrete and continuous problem variables. The continuous model is re-learned after every solution in the population has been sampled. The discrete model however is only re-learned after all of the $2l_d - 1$ nodes of the linkage tree have been processed for all solutions. This balances the discrete and continuous evaluations much better regardless of the ratio between those types of variables.

The algorithm generates the initial population randomly. Each solution $\mathcal{P}_i$ in the population consists of a continuous component $\mathcal{P}_{i_c}$ as well as a discrete component $\mathcal{P}_{i_d}$. $\mathcal{X}_i$

is the offspring solution. The core of the algorithm has two nested loops, which iterate over each linkage tree subset, and additionally iterate over every solution for each of the subsets. The continuous model is learned after a given subset was applied to each solution. The discrete model, however, is learned only after all the subsets have been tried on all solutions. This process continues until a termination condition is reached. New solution acceptance criteria also differ. The continuous model is learned from the top $\tau = 0.35$ fraction of the population, following iAMaLGaM. When continuous variables are sampled, they are always accepted without any other restrictions. The discrete model is built from the entire population. To generate selection pressure, when a mask is applied, the resulting solution is only accepted if it improves, or is equal to the solution following LTGA.

Mixed-Integer Hybrid EA
for $i \in \{0, 1, \ldots, n-1\}$ **do**
  $\mathcal{P}_i \leftarrow$ CREATERANDOMSOLUTION()
  EVALUATEFITNESS($\mathcal{P}_i$)

while $\neg$TERMINATIONCRITERIONSATISFIED **do**
  LEARNDISCRETEMODEL($\mathcal{P}$)
  for $i \in \{0, 1, \ldots, 2l_d - 1\}$ **do**
    $\mathcal{S} \leftarrow$ TRUNCATIONSELECTION($\mathcal{P}, \tau$)
    LEARNCONTINUOUSMODEL($\mathcal{S}$)
    for $j \in \{0, 1, \ldots, n-1\}$ **do**
      $\mathcal{X}_i \leftarrow$ GENERATECONTINUOUSPART($\mathcal{P}_i$)
      $\mathcal{X}_i \leftarrow$ GENERATEDISCRETEPART($j, \mathcal{X}_i, \mathcal{P}$)
  $\mathcal{P} \leftarrow \mathcal{X}$

GENERATECONTINUOUSPART($\mathcal{P}_i$)
  $\mathcal{X}_c \leftarrow$ SAMPLECONTINUOUSMODEL()
  $\mathcal{X} \leftarrow \mathcal{X}_c \cup \mathcal{P}_{i_d}$
  EVALUATEFITNESS($\mathcal{X}$)
  return $\mathcal{X}$

GENERATEDISCRETEPART($j, \mathcal{X}_i, \mathcal{P}$)
  $\mathcal{X}_{prev} \leftarrow \mathcal{X}_i$
  donor $\leftarrow$ GETRANDOMSOL($\mathcal{P}$)
  $\mathcal{X}_d \leftarrow$ COPYSUBSET (j, *donor*, $\mathcal{X}_i$)
  $\mathcal{X} \leftarrow \mathcal{X}_{i_c} \cup \mathcal{X}_d$
  EVALUATEFITNESS($\mathcal{X}$)
  if *fitness*($\mathcal{X}$) $\geq$ *fitness*($\mathcal{X}_{prev}$) **then**
    return $\mathcal{X}$
  else
    return $\mathcal{X}_{prev}$

Fig. 1. Pseudo-code for generating solutions for mixed integer problems with the integrated version of the LTGA and iAMaLGaM Learning Models

4 Experimental Results

4.1 Benchmark Problems

To design the MI benchmarks we use some well-established benchmark problems and adapt them into the MI setting. In all problems, minimization is assumed. Definitions of the well-established benchmark functions can be found in Table 1. Note that due to minimization, zero is the optimal value for all our functions.

The Sphere function is a very simple continuous function, where all variables are completely independent. Rotated Ellipsoid is a stretched version of the Sphere function. The **R** matrix rotates all variables by 45 degrees, creating dependencies between all continuous variables. In the discrete domain, Onemax is arguably the simplest discrete function, with no parameter dependencies. A Deceptive Trap function is a dependent discrete function. The DT5 function we use here is a non-overlapping, additively decomposable composition of the well-known deceptive trap function, with order k=5.

Table 1. Continuous and Discrete functions which are used to define our MI benchmarks

Function Name	Domain	Definition
Sphere	Continuous	$F_{Sphere}(X_c) = \sum_{i=0}^{l_c-1} c_i^2$
Rotated Ellipsoid	Continuous	$F_{R.Ellip.}(X_c) = F_{Ellip.}(\mathbf{R} * X_c)$, where $F_{Ellip.}(X_c) = \sum_{i=0}^{l_c-1} 10^{6*i/(l_c-1)} * c_i^2$
Onemax	Discrete	$F_{Onemax}(X_d) = \sum_{i=0}^{l_d-1} d_i$
Deceptive Trap	Discrete	$F_{DT5}(X_d) = \sum_{i=0}^{l_d/k-1} f_{Trap-k}^{sub}(\sum_{j=ki}^{ki+k-1} d_j)$, where $f_{Trap-k}^{sub} = \begin{cases} 0 & : if\ u = k \\ 1-(k-1-u)/k & : otherwise \end{cases}$

Independently Mixed Benchmarks. We consider all combinations of discrete and continuous problems where the contributions of the discrete and continuous parts are kept independent through addition, see Table 2. Variables in F_1 are fully independent. Only continuous variables are dependent in F_2. Only discrete variables are dependent in F_3. In F_4 both sub-spaces are dependent.

Table 2. $F_1 - F_4$: Domain Independent MI Benchmarks

ID	Function name	Definition
F_1	OnemaxSphere	$F_1(X_d, X_c) = F_{Onemax}(X_d) + F_{Sphere}(X_c)$
F_2	Rotated Ellipsoid	$F_2(X_d, X_c) = F_{Onemax}(X_d) + F_{R.Ellip.}(X_c)$
F_3	DT5Sphere	$F_3(X_d, X_c) = F_{DT5}(X_d) + F_{Sphere}(X_c)$
F_4	DT5Ellipsoid	$F_4(X_d, X_c) = F_{DT5}(X_d) + F_{R.Ellip.}(X_c)$

Cross-Domain Dependence Benchmark. The first four of our proposed benchmark problems keep the dependencies within either continuous, discrete or both parameter sub-spaces. The F_5 benchmark includes cross-domain dependencies between the continuous and discrete variables. It is a specific combination of the previously defined F_{DT5} function with the rotated ellipsoid. It is additively decomposable and consists of sub-functions pertaining to blocks of k discrete and k continuous variables.

More specifically, for a trap function with $k = 5$, there are $2^k = 32$ different binary combinations per block. A differently translated rotated ellipsoid function corresponds with each of those combinations (the origin of each function was randomly generated in [-5,5]). This way, the continuous function which is being optimized depends on the binary counterpart, introducing dependencies between the discrete and continuous variables that pertain to the same subset. In this benchmark the number of discrete variables is the same as continuous variables: $l_d = l_c = l/2$.

$$F_5(X_d, X_c) = \sum_{i=0}^{0.5l/k-1}(1+10^a f_{sub}^{trap}(\sum_{j=ki}^{ki+k-1} d_j))*(1+f_{Ellipse}^{sub}(D_i^{block}, C_i^{block})),$$

where D_i^{block} is a block of five discrete variables, and C_i^{block} are the corresponding five real variables. The D block variables determine which of the 2^k different ellipsoid functions need to optimized, while the C block provides the values of the ellipsoid function

variables. The a value acts as a scaling factor, changing the scale of contribution from the trap function to the overall fitness. In order to solve this benchmark, the algorithm needs to not only solve the trap function, but also optimize the correct rotated ellipsoid function.

4.2 Results on Domain Independent Problems

Heat Maps. We compute the population size that corresponds to the minimal total evaluations needed to solve a problem. The results shown are based on the population sizes for which each problem was solved with the precision of 10^{-10} at least 29/30 times with least evaluations. The results were obtained via bisection.

For the heat maps and scalability analysis of $F_1 - F_4$, we consider different problem lengths: 40, 80, 120 and 160 total variables. For each of these problem sizes, we consider different proportions of variables used with 5, $.25l$, $.5l$, $.75l$ and $l - 5$ continuous variables (and $l - l_c$ corresponding discrete variables).

The heat maps in Figure 2 show how the proportions of variable types (discrete or continuous) affects algorithmic efficiency in terms of population sizes and evaluations required to solve the problem.

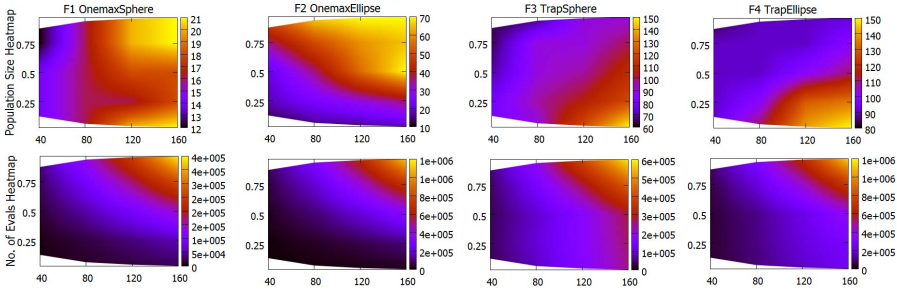

Fig. 2. Heat Maps representing the population sizes (top row) and evaluations (bottom row) needed for different variable compositions. Horizontal axis represents the problem length, the vertical axis is the fraction of continuous variables ($l_c/(l_c + l_d)$) in the problem.

Intuitively F_1 should be the simplest problem for the algorithm to handle, as it contains no parameter dependencies. As the problem composition shifts towards more continuous landscape, the algorithm requires more evaluations. The required population sizing is less affected by the problem composition for F_1 then for the remaining benchmarks. This can be explained by the simplicity and independence of all problem variables.

The effects of changing the problem composition strengthen when partial dependencies are introduced into the problem landscape. As with F_1, for the remaining benchmarks more evaluations are also required for the same problem sizes as the composition of the problem shifts towards larger numbers of continuous variables. Moreover, as expected, benchmarks which contain dependencies within the continuous sub-space, F_2 and F_4, require larger number of evaluations than F_1 or F_3.

Population sizes are also affected by the problem composition. In F_3 and F_4 we observe much larger population size requirements, as the landscape of these functions includes discrete variable dependencies.

This shows that in addition to problem length, the composition of the problem and variable dependencies are a big factor for efficiency in terms of evaluations and population sizes.

Scalability. Figure 3 demonstrates changes in scalability of population size and evaluations over benchmarks $F_1 - F_4$ when the proportions of discrete to continuous variables is changed. Results are shown on a log-log scale. This means that polynomial scalability is indicated by straight lines. From the scalability graphs it is clear that factors such as variable ratios and dependencies can strongly affect the behavior of our algorithm. As expected, the results exhibit polynomial scalability on the tested MI problems.

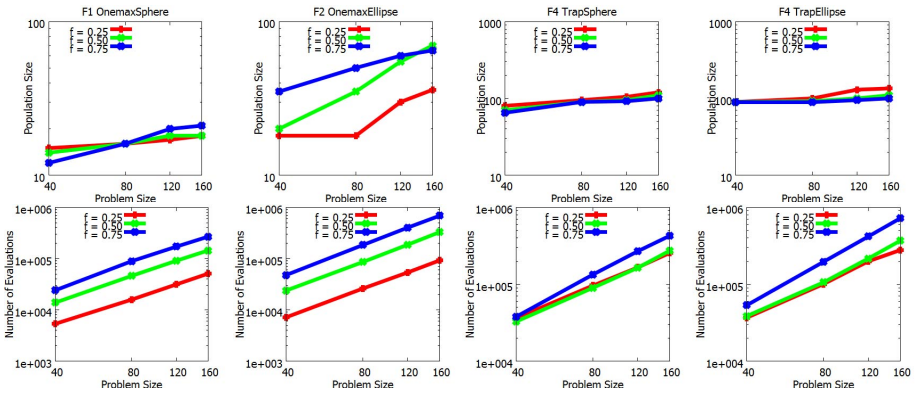

Fig. 3. Scalability of population size and evaluations required for benchmarks $F_1 - F_4$. Fraction f represents the fraction of continuous variables in the problem with length l.

Table 3 shows linear least squares regressions on log-log-scaled data for minimal average number of evaluations e and corresponding population sizes n depending on the problem length l and error term ϵ as follows:

$$log(n) = log(l^{\alpha}) + \epsilon \quad \text{and} \quad log(e) = log(l^{\beta}) + \epsilon.$$

Table 3 shows that population sizes scale sub-linearly. It also shows that performance in terms of evaluations scales more favorably as the problem shifts towards more discrete variable composition.

The use of incremental estimates of iAMaLGaM is crucial in keeping small population sizes. Maintaining evaluation balance and asynchronous model learning, as described in the earlier section, leads to preventing over-exploration or premature convergence much more efficiently than learning the continuous and discrete models synchronously. Graphs (a) and (b) in Figure 4 verify these observations by showing the improved scalability on F_4 over an approach using the original AMaLGaM and with the generations of both models advancing simultaneously.

Table 3. Regression coefficients for scalability

l_c / l_d	$\mathbf{F_1}$ α	$\mathbf{F_2}$ α	$\mathbf{F_3}$ α	$\mathbf{F_4}$ α	$\mathbf{F_1}$ β	$\mathbf{F_2}$ β	$\mathbf{F_3}$ β	$\mathbf{F_4}$ β
5 / l-5	0.3089	0.1621	0.4458	0.3840	1.1420	1.0663	1.4376	1.4132
0.25 l / 0.75 l	0.1286	0.5216	0.2815	0.3146	1.6284	1.8432	1.4017	1.4876
0.5 l / 0.5 l	0.1952	0.9152	0.3089	0.1370	1.6972	1.8940	1.5187	1.6252
0.75 l / 0.25 l	0.4218	0.4557	0.3021	0.0727	1.7456	1.9291	1.7572	1.8815
l-5 / 5	0.3307	0.3307	0.2620	0.0539	1.8381	2.0427	1.8566	1.9880

Some overhead still exists, however. By simply combining a continuous problem with a discrete one into one population and treating it as one MI problem, evaluation overhead is introduced into the algorithm. In graph (b) of Fig. 4, the "Separate" label demonstrates what happens if the discrete and continuous sub-domains are solved separately. This separate approach is more efficient because the optimal population sizes can be chosen individually. Additionally, the fitness function is not affected by the other sub-domain. Of course this separate approach is not possible if cross-domain dependencies are present in the problem.

We also consider an alternative approach based on Evolution Strategies (ES). MIES [5] is an ES which extends $(\mu + \lambda)$-ES for continuous problems to the mixed-integer search spaces. This approach applies a recombination operator, followed by a mutation operator for every solution. Those operators are defined differently for continuous, nominal discrete and integer problem variables. This procedure repeats until λ offspring solutions are created. Best μ solutions from the union of the μ parent solutions and the λ offspring are selected and carried over into the population $P(t+1)$ [6].The MIES algorithm has already been shown to work efficiently on some specific industrial problems as well as a set of general MI benchmarks [5]. In graph (c) of Figure 4 we show the performance of MIES on the F_1 benchmark. While it shows good scalability on this simple benchmark, MIES was not able to solve the remaining $F_2 - F_4$ benchmarks within our experimental limits. We conclude that due to existing variable dependencies in $F_2 - F_4$ MIES is not able to solve them efficiently, and requires additional mechanisms in order to handle such MI problems.

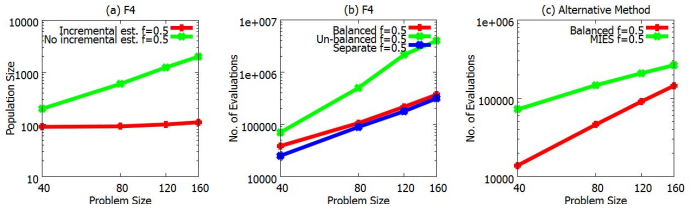

Fig. 4. Importance of (a) incremental estimates and (b) evaluation balancing on F_4. (c) MIES performance on F_1.

4.3 Results on the Dependent Problem

Experimental results on F_5 show that the hybrid algorithm we propose, which integrates two independent models is in fact capable of solving this fully dependent benchmark, however not without encountering another important obstacle which is unique to MI problems, and which did not manifest itself as strongly in the other benchmarks. In this problem, blocks of discrete variables control which continuous function needs to be optimized with counterpart continuous variables. The discrete variables, as well as continuous variables, contribute to the overall fitness. This means that in order to find the global optimum, the best discrete variable assignment has to be found, and the continuous function mapped to this assignment must be optimized. The problem arises in the actual numerical values of fitness contributions from either the discrete or continuous sides. The cumulative fitness value of F_5 can be very deceiving depending on the actual differences in scale of fitness contributions. In a regular, strictly discrete Deceptive Trap function the actual fitness values are irrelevant - as long as the function remains deceptive. This is no longer the case in a mixed-integer setting, where the total fitness relies on the contribution from the continuous domain as well. If the trap values are very small in comparison with the continuous variables fitness contributions it becomes more difficult to optimize the discrete variables as they only appear as irrelevant noise to the evaluation function. On the other hand, if the trap values are significantly larger than ones from the continuous subspace, the problem becomes simpler. This behavior is illustrated in Figure 5.

As shown in the definition of F_5, a controls the scaling of the actual values of the trap function. The larger a, the larger the fitness contribution of the deceptive trap values. In essence, the larger a, the more important it is for the algorithm to solve the trap function of F_5. Figure 5 demonstrates how much impact this factor has on the success of the algorithm. We were not able to consistently solve F_5 for a values $<$1.1. For these values, the trap function fitness contributions are initially very small, resulting in the algorithm prematurely converging on sub-optimal solutions. As a increases, the problem becomes simpler and requires smaller population sizes and less evaluations.

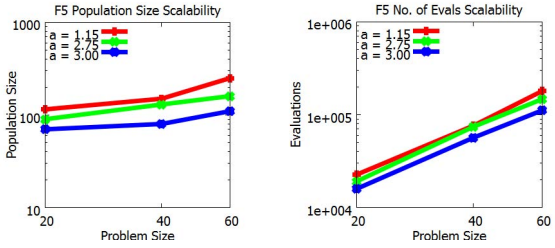

Fig. 5. Scalability on F_5 for different values of a

5 Discussion and Conclusions

Mixed-Integer problems introduce many optimization challenges which do not arise in purely real or discrete optimization problems. With the use of carefully designed

benchmarks, we were able to identify some of such challenges. Obtaining a proper balance in exploration of model information for different types of variables, varying variable ratios and additional overhead or fitness contribution scaling are some of the important issues which should be taken into account when solving MI problems. We made use of two algorithms: LTGA and iAMaLGaM, which are state-of-the-art model-based EAs for problems in discrete and continuous spaces respectively. By extracting key features from these algorithms and carefully integrating the two different models, we were able to study and solve mixed integer benchmarks with varying degrees of variable dependencies. The resulting algorithm achieved polynomial scale-up behavior on the tested benchmarks. We showed that a well-balanced algorithm can solve even very dependent mixed-integer problems, despite having independent model learning methods for the discrete and continuous sub-spaces. The results provide a good foundation and motivation for further work in mixed-integer landscapes with model building EAs. The existing MI EA known as MIES was not able to solve most of our benchmark problems.

While it is very interesting to see that an independent learning approach is capable of solving strongly dependent MI problems, we also learned that this approach has its limitations. Problems with dependencies between the continuous and discrete subspaces can be troublesome to an approach using independent learning models. This was shown on the F_5 where for some values of a the algorithm did not succeed. This motivates further work into fully integrating the discrete and continuous models in order to allow learning of cross-domain dependencies, making it possible to solve a greater range of dependent problems.

References

1. Bosman, P.A.N., Grahl, J., Thierens, D.: Enhancing the Performance of Maximum-Likelihood Gaussian EDAs Using Anticipated Mean Shift. In: PPSN, pp. 133–143 (2008)
2. Bosman, P.A.N., Grahl, J., Thierens, D.: AMaLGaM IDEAs in noiseless black-box optimization benchmarking. In: GECCO (Companion), pp. 2247–2254 (2009)
3. Emmerich, M., Grötzner, M., Groß, B., Schütz, M.: Mixed-Integer Evolution Strategy for Chemical Plant Optimization with Simulators. In: Parmee, I.C. (ed.) Evolutionary Design and Manufacture, pp. 55–67. Springer, London (2000)
4. Emmerich, M.T.M., Li, R., Zhang, A., Flesch, I., Lucas, P.: Mixed-Integer Bayesian Optimization Utilizing A-Priori Knowledge on Parameter Dependences. In: BNAIC 2008, pp. 65–72 (2008)
5. Li, R., Emmerich, M.T.M., Eggermont, J., Bäck, T., Schütz, M., Dijkstra, J., Reiber, J.H.C.: Mixed Integer Evolution Strategies for Parameter Optimization. Evolutionary Computation 21(1), 29–64 (2013)
6. Runarsson, T., Yao, X.: Constrained evolutionary optimization. In: Evolutionary Optimization. International Series in Operations Research and Management Science, vol. 48, pp. 87–113. Springer, US (2002)
7. Thierens, D.: The linkage tree genetic algorithm. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 264–273. Springer, Heidelberg (2010)
8. Thierens, D., Bosman, P.A.N.: Optimal mixing evolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2011, pp. 617–624. ACM, New York (2011)