GAMBIT: A Parameterless Model-Based Evolutionary Algorithm for Mixed-Integer Problems

Krzysztof L. Sadowski k.l.sadowski@uu.nl
Department of Computer Sciences, Utrecht University, Utrecht, The Netherlands

Dirk Thierens d.thierens@uu.nl
Department of Computer Sciences, Utrecht University, Utrecht, The Netherlands

Peter A.N. Bosman peter.bosman@cwi.nl
Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands

Abstract

Learning and exploiting problem structure is one of the key challenges in optimization. This is especially important for black-box optimization (BBO) where prior structural knowledge of a problem is not available. Existing model-based Evolutionary Algorithms (EAs) are very efficient at learning structure in both the discrete, and in the continuous domain. In this paper, discrete and continuous model-building mechanisms are integrated for the Mixed-Integer (MI) domain, comprising discrete and continuous variables.

We revisit a recently introduced model-based evolutionary algorithm for the MI domain, the Genetic Algorithm for Model-Based mixed-Integer opTimization (GAMBIT). We extend GAMBIT with a parameterless scheme that allows for practical use of the algorithm without the need to explicitly specify any parameters. We furthermore contrast GAMBIT with other model-based alternatives. The ultimate goal of processing mixed dependences explicitly in GAMBIT is also addressed by introducing a new mechanism for the explicit exploitation of mixed dependences. We find that processing mixed dependences with this novel mechanism allows for more efficient optimization.

We further contrast the parameterless GAMBIT with Mixed-Integer Evolution Strategies (MIES) and other state-of-the-art MI optimization algorithms from the General Algebraic Modeling System (GAMS) commercial algorithm suite on problems with and without constraints, and show that GAMBIT is capable of solving problems where variable dependences prevent many algorithms from successfully optimizing them.

Keywords

Genetic Algorithms, Estimation-of-Distribution Algorithms, Mixed-Integer Optimization.

1 Introduction

The key to efficient problem optimization lies in efficiently exploiting the problem's structure. The exploitable problem structure, however, may not be directly available. This issue is encountered in various areas of industry, for example, when the exact problem formulation such as the objective function definition, or constraint specifications are not known or provided. Problems where no prior knowledge about the problem structure is available are known as black-box optimization (BBO) problems. This knowledge can be gained, however, by learning useful structural problem features during optimization.

Many real-world applications are based on solving problems which contain both discrete and continuous variables. Such problems are known as mixed-integer (MI) problems and are often regarded as particularly difficult to solve. While many optimization techniques exist about discrete or continuous optimization individually, the area of mixed-integer optimization is relatively less explored. Approaches to mixed-integer optimization do, however, exist. Some of them can be found within the General Algebraic Modeling System (GAMS) framework (Bussieck and Meeraus (2004)). GAMS is a suite of commercial optimizing software, which includes a collection of MI problem solvers. Another alternatives include an extension of the well-known $(\mu + \lambda)$ evolution strategy (ES) for mixed integer problems, MIES (Li et al. (2013)) and the covariance matrix adaptation evolutionary strategy CMA-ES (Hansen (2011)). A Bayesian network based approach has also been introduced with the Mixed-Integer Bayesian Optimization Algorithm (MIBOA) by Emmerich et al. (2008). While we consider a selection of already existing approaches, this article explores yet another alternative: model-based Evolutionary Algorithms (EAs).

There are various ways in which the structure of a problem can be exploited. Through model building, model-based EAs attempt to dynamically capture the problem structure, often by estimating variable dependences. The work on model-based EAs in both the continuous and the discrete domains is extensive, and has been especially successful in black-box settings. The robustness and efficiency that EAs are known for, along with recently published results on combining continuous and discrete model-building EAs, motivates the idea that model-based EAs can be a potentially powerful approach for the MI domain. The main research objective of this paper is to establish whether the capacity of model-based EAs to capture and exploit problem structure extends to the MI domain.

In recent work (Sadowski et al. (2014)) we have introduced the Genetic Algorithm for Model-Based mixed-Integer opTimization (GAMBIT) and suggested that combining model-based EA approaches from the discrete and continuous domains is a promising foundation for dealing with mixed-integer problem landscapes. However GAMBIT, like many algorithms, requires specification of some parameter settings before it can begin any optimization process. Population size, for example, is an essential parameter in evolutionary computation. Research results are often presented using the optimal population size parameter settings, which are determined empirically. While this approach is academically insightful, it may often be very impractical, or even infeasible to determine the optimal population size. In order to eliminate the need to specify a population size, we modify and apply a population size-free scheme proposed by Harik and Lobo (1999)for the so-called parameterless GA to GAMBIT. Additionally, we introduce a mechanism which allows for the removal of another important parameter from the user input in GAMBIT: the number of clusters to be used. Coupled with the population size-free scheme, this results in a practical mechanism which removes the need to specify any GAMBIT parameters by the user. Parameterless GAMBIT refers to the use of this mechanism with GAMBIT.

Previous work on GAMBIT highlights the importance of properly balancing the use of discrete and continuous models. To further motivate the balanced model approach, we extend our previous work on GAMBIT by comparing it to alternative algorithms based on similar model-based approaches, which do not maintain balance between the continuous and discrete models in the same fashion. Instead, they allow for one model to be the driving force of the optimization process. Specifically, we introduce an algorithm which uses discrete model building techniques only, by discretizing the continuous space. Conversely, we consider an algorithm using only continuous model-building techniques by treating discrete variables as continuous. Additionally we consider an algorithm which couples a model-building approach for a continuous domain with a local search algorithm for the discrete domain.

MI problems may include variable dependences between the discrete and continuous problem variables. So far, GAMBIT addressed such dependences only indirectly, with the use of a clustering mechanism explained later in this article. One of the biggest challenges in MI optimization however, is the ability to directly exploit such mixed dependences. We introduce a novel mechanism to process mixed variable dependences, and integrate it into GAMBIT. We wish to learn whether sampling models with mixed dependences has the potential to result in more efficient optimization. To accomplish this, we provide a priori configured mixed models to GAMBIT and study if the mixed dependency processing mechanism can cause a significant decrease in evaluations needed to solve benchmark problems, or if it will result in additional overhead.

Many state-of-the-art industrial mixed-integer optimization software tools exist that are efficient and successful on a wide range of constrained and unconstrained problems. However, when faced with challenging non-convex problems they may not perform as well. We therefore contrast our work with state-of-the-art MI optimization algorithms included in the previously mentioned General Algebraic Modeling System (GAMS) (Bussieck and Meeraus (2004)) commercial algorithm suite as well as the Mixed-Integer Evolution Strategy (MIES) (Li et al. (2013)).

The remainder of this paper is structured as follows. Section 2 provides background information on the model-building evolutionary algorithms LTGA and iAMaLGaM, designed for discrete and continuous problems variables respectively. Section 3 details GAMBIT and its components. Section 4 introduces the Mixed-Integer benchmarks and explains their landscape features. In Section 5 we introduce and test a parameterless implementation of GAMBIT. Section 6 summarizes the algorithms that extend known model-based approaches into the MI space and compares their performance. Section 7 introduces a new sampling mechanism to GAMBIT, which allows for processing and sampling of mixed dependences. Section 8 provides a comparison to a set of commercial solvers on our benchmark set. Finally, Section 9 is a discussion section and Section 10 concludes this article and motivates future work.

2 Background on two related model-building EAs: LTGA and iAMaLGaM

Model-based EAs have a specifically identifiable model that governs how they perform optimization. By learning how to configure this model on-line, i.e. during optimization, model-based EAs can capture and exploit problem structure, which can be particularly useful in the black-box setting. GAMBIT combines mechanisms from known model-based EAs for the discrete and continuous optimization domains to perform optimization in the mixed-integer domain. Specifically, GAMBIT combines the Linkage Tree Genetic Algorithm (Thierens (2010)) from the discrete, and iAMaLGaM (Bosman et al. (2008)) from the continuous domain. Both LTGA and iAMaLGaM are model-based EAs which have been proven to be competent and efficient approaches in their respective domains [1].

[1]The source code of both LTGA and iAMaLGaM is available for download online at `http://homepages.cwi.nl/~bosman/source_code.php`

2.1 LTGA

The Linkage Tree Genetic Algorithm (LTGA) is a member of the Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) family (Thierens and Bosman (2011)). For any GOMEA instance, to generate a new solution, a collection of subsets of problem variables, known as the Family of Subsets, or FOS, is traversed and each subset is used in a mixing operation with existing solutions. The rationale of the FOS structure is to identify and exploit subsets which make up important building blocks. Generally, FOS subsets could be composed of any number of variables. To determine the subsets, the specific GOMEA instance LTGA employs hierarchical clustering to arrive at a linkage tree that expresses a clustering of variable dependences, ranging from independent to fully dependent. Each node of this tree is considered a FOS subset. An example of a linkage tree and the associated FOS is given in Figure 1.

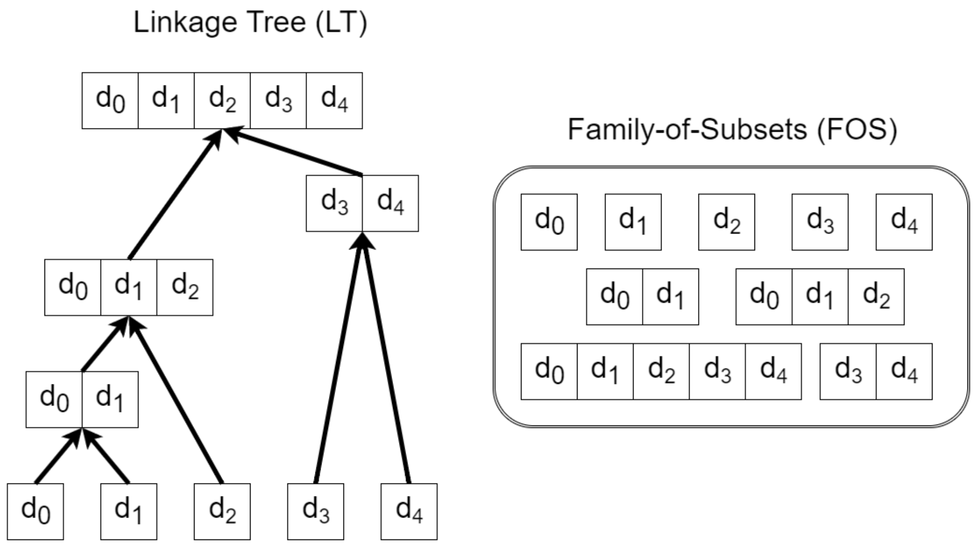

Figure 1: A linkage tree created with a hierarchical clustering algorithm, and its associated FOS structure.

Specifically, in each generation of LTGA, hierarchical clustering is performed. To this end mutual information is used as the basis for a distance measure. A tree is built from the bottom-up, where each node represents a subset of problems variables. This process starts from singleton clusters that are leaf nodes of the linkage tree, where each cluster contains only one problem variable. Each linkage tree node represents a learned dependency between some subset of problem variables. As illustrated in Figure 1, each linkage tree node becomes a subset of the FOS. Using the linkage tree as a FOS structure allows for learning important building blocks of the solution while retaining good computational efficiency, as exactly $2l_d - 1$ subsets are created each time, where l_d represents the number of discrete variables.

The general procedure of LTGA is as follows. First, an initial population of N randomly generated solutions is created and each solution is evaluated. In each generation, the linkage tree is built using hierarchical clustering. Next, LTGA iterates over every solution in the population. For each solution, a copy is created and represents an offspring solution and LTGA iterates over each of the subsets in the FOS. For each of these subsets, a random donor solution from the current population is selected. Variable values specified by the current FOS subset of the donor solution are copied onto the corresponding variables of the current offspring solution. This mixing process alters the offspring solution, which is immediately evaluated. If this mixing, depicted in Figure 2, results in an equally good or better solution than the solution before mixing, the new offspring solution is saved. Otherwise the changes are discarded. LTGA then moves on to the next subset of FOS and the mixing process is applied again to the offspring solution. If no improvement is found after processing all of the FOS subsets a Forced Improvements (FI) mechanism is invoked. FI repeats the mixing procedure, using as the donor the best solution currently available in the population. FOS subsets are processed again and once an improvement is found for a subset, the solution is accepted and the remaining subsets are omitted. FI have been shown to improve the convergence of solutions, without overly increasing the selection pressure (Bosman and Thierens (2012)). After each solution in the population is processed in this fashion, the offspring solutions replace the parent population, the algorithms proceeds to a new generation, and the process begins anew.

LTGA achieves very good results on problems such as Deceptive Trap, NK-Landscapes, MaxCut, and Hierarchical Functions (Bosman and Thierens (2012)) as well as MAX-SAT (Sadowski et al. (2013)) .

2.2 iAMaLGaM

The Incremental Adapted Maximum-Likelihood Gaussian Model Iterated Density Estimation Evolutionary Algorithm (iAMaLGaM) is a state-of-the-art Estimation-of-Distribution Algorithm (EDA) for real-valued black-box optimization (Bosman et al. (2008)). Following the general EDA paradigm, iAMaLGaM estimates a Gaussian distribution every generation from the selected solutions and generates new solutions by sampling the estimated distribution. The mean vector and covariance matrix are estimated incrementally using intergenerational memory decay on maximum-likelihood estimates. A mechanism which scales up the co-variance matrix when needed is used to counteract the risk of premature convergence on slope-like regions of the search space. Finally, the intergenerational Anticipated Mean Shift procedure improves the behavior of iAMaLGaM in slope-like regions of the search space. iAMaLGaM has been shown to achieve rotation invariance and very good scale-up on many well-known black-box optimization benchmarks (Bosman et al. (2009)).

3 GAMBIT

The Genetic Algorithm for Model-Based mixed Integer opTimization (GAMBIT) is an integration of LTGA and iAMaLGaM for mixed-integer problems [2]. Components of GAMBIT were first described by Sadowski et al. (2014) and showed promising results for MI optimization. The algorithm was formally introduced as GAMBIT in the follow-up work, which also introduced a clustering mechanism and constraint handling techniques, allowing GAMBIT to solve a wider range of problems, including ones which require symmetry-breaking and the handling of constraints (Sadowski et al. (2015)). This section compiles and summarizes the most up-to-date version of GAMBIT.

Optimization with GAMBIT is done in two parts. First, at the beginning of each generation, a clustering mechanism splits the current population into k subpopulations. This clustering allows for more efficient optimization of multi-modal

[2] Executable code of the version of GAMBIT presented in this paper is available at `https://svn.science.uu.nl/repos/sci.3611914.SO-GAMBIT`

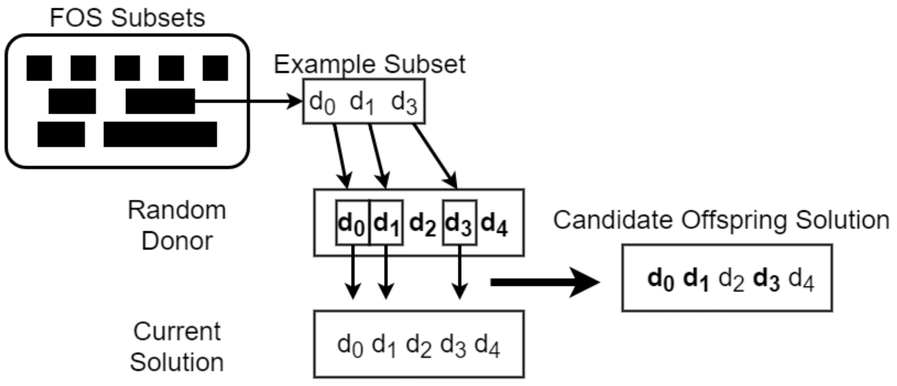

Figure 2: Creating a candidate solution with LTGA

problems, and was shown to improve GAMBIT's performance on problems with mixed variable dependences (Sadowski et al. (2015)). Optimization is performed with instances of an integrated model-based mechanism on each sub-population. The offspring solutions from each instance are joined, replacing the previous population, and the process begins anew. A basic overview of the algorithm is presented in Figure 3. A detailed explanation of the clustering process, and the integrated model building and sampling are presented next.

```
GAMBIT Overview
$\mathcal{P} \leftarrow$ GenerateAndEvaluateRandomPopulation($n$)
while $\neg$TerminationCriterionSatisfied do
    $\mathcal{C} \leftarrow$ DetermineClusterCenters($\mathcal{P}, k$)
    for $j \in \{0, 1, \ldots, k\}$ do
        $\mathcal{P}_k \leftarrow$ DetermineSubPopulation($\mathcal{P}, \mathcal{C}_k$)
        $\mathcal{O}'_k \leftarrow$ IntegratedMechanism($\mathcal{P}_k$)
    $\mathcal{P} \leftarrow \mathcal{O}'_0 \cup \mathcal{O}'_1 ... \cup \mathcal{O}'_k$
```

Figure 3: Pseudo-code overview of GAMBIT. $\mathcal{P}$ represents the population of size n. $\mathcal{C}$ is a set of solutions representing cluster centers for k clusters, or sub-populations. $\mathcal{P}_i$ represents the i-th sub-population and $\mathcal{O}'_i$ is the i-th set of offspring generated from sub-population i.

3.1 Clustering Mechanism

The clustering process takes place at the beginning of every generation, where k cluster centers are first selected. The best solution available in the current population is first selected as a cluster center. Remaining cluster centers are iteratively selected as solutions with the furthest distance from all already chosen centers. Once the centers are determined, n/k solutions closest to the chosen centers are added to the clusters. This approach to determining clusters is based on a successful approach previously applied to multi-objective optimization (Rodrigues et al. (2014)).

Previous work suggests that a mixed distance metric between solutions leads to the best performance of GAMBIT. Specifically, the distance between two solutions is captured by scaling down the continuous variables to a 0-1 range, where the largest value in the population takes on the value of one, and the smallest zero. Binary variables are then treated as real values and the Euclidean distance is computed between two solutions.

Each cluster represents a (possibly overlapping) sub-selection of solutions from the population. For each of these sub-populations, the integrated mechanism is used to generate offspring for a given generation.

3.2 Integrated-Models Mechanism

The key to efficient performance of GAMBIT relies on a properly balanced model-building and sampling mechanism, which is used for every sub-population.

Recall that LTGA uses subsets of variable indices generated by learning a linkage-tree over the discrete variables, while iAMaLGaM samples all the continuous variables for a given solution at once to generate offspring solutions. GAMBIT aims to preserve both of these mechanisms. Two key modifications are made to the FOS mixing process in order to extend it for continuous subsets and allow for interleaving of discrete and continuous models.

The first key modification allows for the inclusion of continuous subsets within the FOS. Recall that LTGA builds a linkage-tree with $2l_d - 1$ nodes representing important subsets of discrete variables. These FOS subsets are iterated over for each solution, and offspring candidates are generated. To keep a proper balance between sampling of the discrete and continuous variables, and simultaneously account for the difference in problem composition (i.e. number of discrete vs. continuous variables in the problem), $2l_c - 1$ continuous subsets are added into the FOS. In order to preserve iAMaLGaM's way of sampling new continuous variables, all continuous subsets added to the FOS in GAMBIT contain all of the continuous problem variables. With the extended version of the FOS, GAMBIT employs two different ways to manage subsets of the FOS. If a subset is a discrete one, the corresponding discrete variables are copied from a random donor solution from the population onto the current solution, following the LTGA procedure. If the subset is of the continuous type, the offspring solution is generated by sampling all of the continuous problems variables from iAMaLGaM's continuous model.

The second key modification is the order in which the FOS subsets are processed. LTGA, for each solution, iterates and processes every FOS subset. This mechanism is modified in GAMBIT, in order to allow for a better balance of interleaving the use of discrete and continuous models. With GAMBIT, the FOS subsets are put in a random order first. Then, one subset is used to generate offspring from all solutions, before moving on to another subset. If it is a continuous subset, after iterating over all solutions and creating new offspring following iAMaLGaM, the continuous model is updated. If it is a discrete subset, the discrete model is only updated after all other discrete subsets have been processed in this generation, following LTGA procedure. This mechanism enables GAMBIT to process discrete and continuous subsets and create offspring solutions in a interleaved fashion, but re-builds and updates the discrete and continuous models at different rates, preserving the original model building properties of LTGA and iAMaLGaM.

The details of the integrated mechanism which makes up the core of GAMBIT are presented in pseudo-code in Figure 4 and are explained here. After the clustering mechanism creates k sub-populations, each of these sub-populations is independently processed by an instance of the integrated mechanism for a single generation. Each of the integrated mechanisms consists of a discrete and a continuous model, and is used to create offspring solutions for a given generation. At the beginning of each generation, the discrete model of the instance is updated and a linkage tree is built over all discrete variables, and used to generate the $2l_d - 1$ discrete FOS elements as illustrated in Figure 1. The FOS is then appended with $2l_c - 1$ continuous subsets, each containing all l_c continuous variables. The FOS subsets are then exhaustively processed in a randomized order. The processing of the subsets depends on the subset type. If the subset type is continuous, the continuous model is updated, and each offspring solution is created by copying all discrete variables from the current parent solution, and sampling all new continuous variables. The offspring solution is always accepted, following the iAMaLGaM sampling procedure. Updating the continuous model consists of re-building a covariance matrix, based on $\tau = 0.35$ fraction of best solutions in the population, and updating iAMaLGaM's intergenerational mechanisms. If the subset is discrete, the variables specified by the subset are copied from a donor solution randomly selected from the current population, and copied onto the current solution, while the remaining discrete variables and all of the continuous variables remain unchanged. This offspring solution is only accepted if it is not worse than the parent solution. This procedure is illustrated in Figure 2. After every subset is processed for every solution, the generation ends. The offspring solutions generated by each instance of the integrated mechanism are merged together, forming a new offspring population, and is used for clustering in the next generation.

3.3 Constraint Handling

MI problems are often accompanied by constraints which limit the feasibility of the search space. In a black-box setting, we cannot make any assumptions about the number or types of constraints present in a given problem. The only feedback we may hope for is an indicator of how strongly the current problem constraints are violated. For such conditions, our previous work (Sadowski et al. (2015)) suggests the use of the Dynamic Penalty Function Method (Runarsson and Yao (2002)).

The Dynamic Penalty Method worsens the value of the objective function if the solution is infeasible. This penalty factor is proportional to the constraint violation value as well to the number of generations which have already passed. This means that as time passes, the penalty factors of infeasible solutions becomes stronger. Because of these features, the penalty method initially allows for exploration of promising but not feasible regions. As time progresses however, feasible regions become more important. In our implementation the penalty function value is the square of the total constraint violation value multiplied by the number of generations which have already passed.

4 Benchmark Problems

In the remainder of this article we shall present extensions of GAMBIT and show their impact on performance directly. In this section we define problems that we use in our experiments.

A Mixed-Integer Problem is defined as follows:

$$min\ f(\boldsymbol{x_d}, \boldsymbol{x_c})$$
$$\text{s.t. } \mathbf{h}(\boldsymbol{x_d}, \boldsymbol{x_c}) = 0, \ \ \mathbf{g}(\boldsymbol{x_d}, \boldsymbol{x_c}) \leq 0$$

where x represents the solution

$$x = \boldsymbol{x_d}\boldsymbol{x_c} = d_0 ... d_{l_d-1}\ c_0 ... c_{l_c-1}$$

where $d_i \in \{0,1\}, c_i \in \mathbb{R}$ and $\boldsymbol{x_d}$, $\boldsymbol{x_c}$ are the groups of all discrete and continuous variables, respectively. f is the objective function. $\mathbf{h}$ and $\mathbf{g}$ are the sets of equality and

```
Integrated Model Sampling for a cluster sub-population $P_k$

INTEGRATEDMODELMECHANISM($\mathcal{P}_k$)
  $\mathcal{S}ubsetType = \{'discrete'\}^{(2*l_d-1)} \cup \{'continuous'\}^{(2*l_c-1)}$
  $\mathcal{S}ubsetType = shuffle(\mathcal{S}ubsetType)$
  UPDATEDISCRETEMODEL($\mathcal{P}_k$)
  for $i \in \{0, 1, \ldots, (SubsetType.length - 1)\}$ do
      if $\mathcal{S}ubsetType(i)$ = 'continuous' then
          $\mathcal{S} \leftarrow$ TRUNCATIONSELECTION($\mathcal{P}_k, \tau$)
          UPDATECONTINUOUSMODEL($\mathcal{S}$)
          for $j \in \{0, 1, \ldots, n-1\}$ do
              $\mathcal{O}_{ki} \leftarrow$ GENERATECONTINUOUSPART($\mathcal{P}_{ki}$)
      else if $\mathcal{S}ubsetType(i)$ = 'discrete' then
          for $j \in \{0, 1, \ldots, n-1\}$ do
              $\mathcal{O}_{ki} \leftarrow$ GENERATEDISCRETEPART($j, \mathcal{O}_{ki}, \mathcal{P}_k$)
      $\mathcal{P}_k \leftarrow \mathcal{O}_k$
  return $\mathcal{P}_k$

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

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

Figure 4: Pseudo-code for generating solutions for mixed integer problems with GAMBITs Learning Models.

inequality constraint functions respectively. If both of these sets are empty, the Mixed-Integer problem is considered to be unconstrained.

4.1 Unconstrained Problems

We first consider a set of unconstrained mixed-integer benchmarks. These benchmarks are designed from well-known discrete and continuous benchmark problems, and are combined in ways that capture different mixed-integer landscape features. Table 1 specifies the discrete and continuous functions used to generate our mixed-integer benchmarks set. This set is designed to test algorithms coping with the following landscape features:

- Intra-domain variable dependences
- Inter-domain variable dependences
- Multiple optima

Intra-domain dependences refer to dependences between variables of the same type. Inter-dependences refer to dependences across the discrete and continuous domains. The first mixed problem F_1 is the Onemax-Sphere function. It is a concatenation of two problems where all the variables are completely independent. F_2 introduces intra-domain dependences due to 45 degree parameter rotation of the continuous problem variables, while the discrete variables remain independent. A counterpart to this function is F_3, where continuous variables are independent, however discrete variables exhibit strong dependences due to the Deceptive Trap (DT) function. DT is a binary decomposable function which is composed of m trap functions. In our problems we consider trap functions of size 5. Its definition in Table 1 shows that the global optimum occurs when all the bits are 1's. However, $2^m - 1$ deceptive local optima exist when all bits are 0's. The DT5-R.Ellipse, or F_4, is a problem where discrete variables are strongly dependent on each other because of the Deceptive Trap Function behavior, while the continuous variables are strongly dependent on each other due to a parameter rotation of 45 degrees. However, no dependence between the discrete and continuous domains exist. Cross-Dependent function F_5 includes intra-domain, as well as inter-domain dependences. 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. For a trap function with $k = 5$, there are $2^k = 32$ different binary combinations per block. A differently translated k-variable 45 degree rotated ellipse function corresponds with each of those combinations. The continuous function which is being optimized depends on the binary counterpart, introducing dependences between the discrete and continuous variables that pertain to the same subset. In the function definition, D_i^{block} is a block of five discrete variables. C^{block} are the corresponding five real-valued variables. The D^{block} variables determine which of the 2^k different rotated ellipsoid functions are being optimized, while C^{block} provides the values of the ellipsoid function variables. In this benchmark the number of discrete variables is the same as the continuous variables: $l_d = l_c = l/2$. Finally, Paired-MixDependency, or F_6 function, is a problem with mixed dependences between variable pairs, where each continuous variable has a dependence with a corresponding discrete variable. All these MI problems are summarized in Table 2.

Table 1: Continuous and discrete functions used to define our unconstrained mixed benchmarks

Functions	**Domain**
$F_{Sphere}(\boldsymbol{x_c}) \;= \sum_{i=0}^{l_c-1} c_i^2$	Continuous
$F_{R.Ellip.}(\boldsymbol{x_c}) \;= F_{Ellip.}(\mathbf{R} * \boldsymbol{x_c})$, where $F_{Ellip.}(\boldsymbol{x_c}) \;= \sum_{i=0}^{l_c-1} 10^{6*i/(l_c-1)} * c_i^2$	Continuous
$F_{Onemax}(\boldsymbol{x_d}) = \sum_{i=0}^{l_d-1} d_i$	Discrete
$F_{DT5}(\boldsymbol{x_d}) \;= \sum_{i=0}^{l_d/k-1} f_{Trap-k}^{sub}(\sum_{j=ki}^{ki+k-1} d_j)$, where $f_{trap\ k}^{sub}(u) = \begin{cases} 0 & : if\ u = k \\ 1 - (k-1-u)/k & : otherwise \end{cases}$	Discrete

4.2 Constrained Benchmarks

Many industrial mixed continuous-discrete problems include constraints. We therefore also consider a set of mixed-integer problems from the MINLP Benchmark Library specified by Bussieck and Pruessner (2003). The specification of the objective function, constraints and parameter ranges can be found in Table 3. This benchmark set con-

Table 2: Unconstrained benchmark problems

ID	Definition	Landscape Feature
F_1	$F_1(\boldsymbol{x_d}, \boldsymbol{x_c}) = F_{Onemax}(\boldsymbol{x_d}) + F_{Sphere}(\boldsymbol{x_c})$	All Independent
F_2	$F_2(\boldsymbol{x_d}, \boldsymbol{x_c}) = F_{Onemax}(\boldsymbol{x_d}) + F_{R.Ellip.}(\boldsymbol{x_c})$	Continuous Dependences
F_3	$F_3(\boldsymbol{x_d}, \boldsymbol{x_c}) = F_{DT5}(\boldsymbol{x_d}) + F_{Sphere}(\boldsymbol{x_c})$	Discrete Dependences
F_4	$F_4(\boldsymbol{x_d}, \boldsymbol{x_c}) = F_{DT5}(\boldsymbol{x_d}) + F_{R.Ellip.}(\boldsymbol{x_c})$	Cont. and Disc. Dep.
F_5	$F_5(\boldsymbol{x_d}, \boldsymbol{x_c}) = \sum_{i=0}^{0.5l/k-1}(1 + 10 f_{trap}^{sub}(\sum_{j=ki}^{ki+k-1} d_j))$ $*(1 + f_{Ellipse}^{sub}(D_i^{block}, C_i^{block}))$	Cont., Disc. and Mix. Dep.
F_6	$F_6(\boldsymbol{x_d}, \boldsymbol{x_c}) = 2 * \sum_{i=0}^{l_c-1}(d_i - c_i^2) - d_i$	Mixed Pair Dependences

tains the same problems as used by Li et al. (2013) for testing the Mixed Integer Evolution Strategy (MIES). Minimization is assumed for both unconstrained and constrained problems.

Table 3: Specifications of the MINLP selected constrained benchmark problems, and initialization ranges for problem variables

Name	Function	Constraints	Range
F_7	$2r_1 + d_1$	$-r_1^2 - d_1^2 \leq -1.5, r_1 + d_1 \leq 1.6$	$r_1 \in [0, 1.6]$ $d_1 \in \{0, 1\}$
F_8	$2r_1 + 3r_2 + 1.5d_1 + 2d_2 - 0.5d_3$	$r_1^2 + d_1 = 1.25, r_2^{1.5} + 1.5d_2 = 3$ $r_1 + d_1 \leq 1.6, 1.333r_2 + d_2 \leq 3$ $-d_1 - d_2 + d_3 \leq 0$	$r_{1,2} \in [0, 10]$ $d_{1,2,3} \in \{0, 1\}$
F_9	$0.8 + 5(r_1 - 0.5)^2 - 0.7d_1$	$-exp(r_1 - 0.2) - r_2 \leq 0$ $r_2 + 1.1d_1 \leq -1, r_1 - 1.2d_1 \leq 0$	$r_1 \in [0.2, 1]$ $r_2 \in [0.22554, -1]$ $d_1 \in \{0, 1\}$
F_{10}	$6 + (r_1 - 1)^2 + (r_2 - 2)^2$ $+(r_3 - 3)^2 - d_1 - 3d_2 - d_3$ $-0.693147180559945d_4$	$r_1 + r_2 + r_3 + d_1 + d_2 + d_3 \leq 5$ $r_1^2 + r_2^2 + r_3^2 + d_3 \leq 5.5$ $r_1 + d_1 \leq 1.2, r_2 + d_2 \leq 1.8$ $r_3 + d_3 \leq 2.5, r_1 + d_4 \leq 1.2$ $r_2^2 + d_2 \leq 1.64, r_3^2 + d_3 \leq 4.25$ $r_3^2 + d_2 \leq 4.64$	$r_{1,2,3} \in [0, 10]$ $d_{1,2,3,4} \in \{0, 1\}$
F_{11}	$-5r_1 + 3r_2$	$8r_1 - 2r_1^{0.5}r_2 + 11r_2 + 2r_2^2 - 2r_2^{0.5} \leq 39$ $r_1 - r_2 \leq 3, 3r_1 + 2r_2 \leq 24$ $r_2 - d_1 - 2d_2 - 4d_3 = 1, d_2 + d_3 \leq 1$	$r_1 \in [1, 10]$ $r_1 \in [1, 6]$ $d_{1,2,3} \in \{0, 1\}$
F_{12}	$(r_4 - 1)^2 + (r_5 - 2)^2 + (r_6 - 1)^2$ $-log(1 + r_7) + (r_1 - 1)^2$ $+(r_2 - 2)^2 + (r_3 - 3)^2$	$r_1 + r_2 + r_3 + d_1 + d_2 + d_3 \leq 5$ $r_6^2 + r_1^2 + r_2^2 + r_3^2 \leq 5.5, r_1 + d_1 \leq 1.2$ $r_2 + d_2 \leq 1.8, r_3 + d_3 \leq 2.5, r_1 + d_4 \leq 1.2$ $r_5^2 + r_2^2 \leq 1.64, r_6^2 + r_3^2 \leq 4.25$ $r_5^2 + r_3^2 \leq 4.64$ $r_4 - d_1 = 0, r_5 - d_2 = 0$ $r_6 - d_3 = 0, r_7 - d_4 = 0$	$r_{1,2,3} \in [1, 10]$ $r_{4,5,6,7} \in [0, 1]$ $d_{1,2,3,4} \in \{0, 1\}$

5 Parameterless Scheme

When algorithmic results are presented, they are often associated with certain parameter settings that were used during execution. For some parameter settings a given algorithm can perform much better than for others on a given problem. Up to this point GAMBIT needed the specification of two crucial parameters: population size and number of clusters used. In this section we explain why the need for setting these parameters can be problematic, especially in a black-box optimization setting, and propose a methodology that allows for a parameterless execution of GAMBIT.

Population size is an essential parameter of evolutionary algorithms. Determining a population size that results in efficient performance, however, is often difficult. Setting the population size too small will result in premature convergence and inability to solve the problem. Choosing the population size too large may result in an unnecessary overhead leading to inefficient performance in terms of evaluations needed. Research on population-based algorithms very often reports results based on empirical data obtained by an algorithm with an optimal population size for a given problem. While the use of bisection to find the optimal population size is very useful to examine algorithmic performance, it is very often infeasible in practice. For example, if the optimum of a problem is not known, it may be impossible to know if a given population size results in good or poor performance. Additionally, to perform a bisection, a stopping criterion is needed when the problem is not being solved for a given population size.

Predicting a good population size for a given problem in a black-box setting can be near impossible. While it is generally true that when the dimensionality of a problem increases, the population size required to solve it also increases, there is very little information that tells us what this optimal population size might be for different problems. This is because many factors affect problem difficulty such as multi-modality, variable interactions or constraints.

Similarly to the population size parameter, determining an optimal number of clusters for a given problem prior to execution may often be impossible in practice. Choosing the wrong number of clusters however, can have a negative impact on the performance. An insufficient number of clusters may lead to the inability to exploit more complex problem structures, while too many clusters will likely result in significant overhead.

In order to design an algorithm which may be useful in a real-world setting, the problem of determining values for the population size and the number of clusters parameters needs to be addressed. We adapt the Parameter-Free scheme initially proposed for discrete optimization by Harik and Lobo (1999). Our goal is to determine if it is possible to adapt this scheme to GAMBIT while retaining similar performance in terms of scalability as with the optimal population size. We test the performance, with and without the parameterless scheme, of GAMBIT on problems with different landscape features and with different problem composition.

5.1 Methodology

The population-free scheme was initially proposed for discrete optimization. In recent work this approach was successfully used with LTGA (Luong et al. (2015)). In this scheme, the optimization algorithm is initialized dynamically over time for different population sizes. The generational progress of each instance is inversely proportional to the instance's population size, allowing for an evaluation balance between instances. Additional mechanisms exist which determine when to terminate smaller instances based on their relative performance or when to start new, larger instances. The general idea behind this scheme is to give instances with smaller population sizes a chance to solve a problem, but allow larger instances to attempt to solve it simultaneously without the need of instances with smaller population sizes to reach a termination criterion.

More specifically, an instance of an algorithm is first created given a minimal population size. This instance runs for b generations, where b represents the base, suggested to be 2 or 4. After this, a new instance of the algorithm is created, with a doubled population size. This instance will perform a single generational step at the frequency $1/b$ smaller than the previous instance. After b generational steps of this instance (by which time the first instance made b^2 generational steps), again a new instance with a double population size is created and the process continues.

A major drawback of using bisection is the need to specify when the algorithm should move on to a larger population size. Such stopping criteria are very often difficult, if not impossible, to specify. With the population-free approach, this is no longer necessary as algorithm instances with larger population sizes will be utilized, alongside of smaller population size instances.

Some modifications are made to this scheme. If for a given population size instance within the parameterless scheme, the average fitness of solutions selected by GAMBIT is higher than the average fitness of the selected solutions for a smaller population size instance, all smaller population size instances cease execution. This is a way to save needless evaluations, since a larger instance already reached better fitness levels. Additionally, the best current solution out of any instance is always preserved and shared among all instances. In our implementation we use b=4. The minimal, or starting population size is based on iAMaLGaM's population size guidelines, which is $10\sqrt{l_c}$. We adapt this guideline to the mixed-integer case, by setting the initial population size to $10\sqrt{l_c + l_d}$.

Determining the number of clusters is done in a similar manner. Every time a new instance (with the doubled population size) is introduced, the number of clusters for this instance is increased by one from the previous instance. With a linearly growing number of clusters k, and exponentially growing population size n, the number of solutions in each cluster will increase with time, as the number of population members per cluster is n/k. In our implementation the starting cluster size $k = 1$.

5.2 Parameter Free Results

We compare the parameter free GAMBIT with GAMBIT parameterized with the optimal population size determined via bisection. We test benchmarks with different landscape characteristics: F_1, F_2, F_3 and F_4. For each of the problems, different problem sizes and problem compositions are considered. Let ρ describe the ratio $\rho = l_c/(l_c + l_d)$ of the continuous variables in a given problem. For instance $\rho = 1.0$ denotes a problem where all variables are continuous, while $\rho = 0.5$ represents a problem with the same number of discrete and continuous variables. We consider different values for ρ: 1.0, 0.75, 0.5, 0.25, 0.0. The success criterion is finding the optimum in at least 29 out of 30 runs.

Heat-maps in Figure 5 show how many evaluations are needed for problems with different dimensionality and variable composition. As could be expected, as the problem dimensionality increases, so does the number of evaluations required. Additionally, a larger number of evaluations is also needed when the problem composition becomes more continuous (larger values of ρ), or when problems containing discrete dependences (F_3 and F_4) are present.

From the heat-maps it follows that the parameterless GAMBIT obtains a similar behavior as the GAMBIT instance with an optimal population size in terms of evaluations distribution. We examine both approaches further by considering scalability.

Table 4 shows a linear least squares regressions on log-log scaled data for the average number of evaluations e depending on the total problem length $l = l_c + l_d$ with the population-free scheme and optimal population size with an error term ϵ as follows:

$$log(e) = log(l^{\alpha}) + \epsilon.$$

Data provided in Table 4 suggests that the population-free scheme with GAMBIT can achieve very similar scalability as the scalability attained with an optimal population size. While in some cases (eg. $F_3 : \rho = 0.75$) the scalability with optimal population

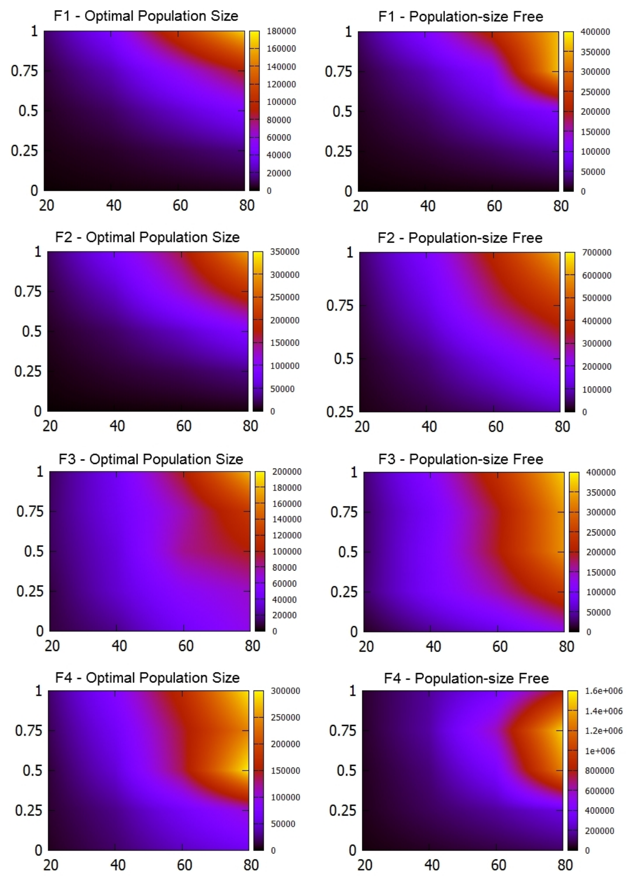

Figure 5: Heat maps representing the number of evaluations needed with the optimal population size and with the parameterless scheme. Horizontal axis represents the problem length, the vertical axis is the fraction of continuous variables ($l_c/(l_c + l_d)$)) in the problem.

size is slightly better, in the majority of cases the population-free scheme scales just as good. In some cases (eg. $F_2 : \rho = 1$) the parameterless scheme even allows for slightly better scaling.

The applicability of an algorithm with the parameterless scheme for many problems where bisection is not feasible, combined with comparable scalability results where an optimal population size can be determined, justifies the use of the parameterless scheme as a preferred alternative. For the remainder of this paper, all GAMBIT results will therefore be obtained using the parameterless scheme, unless otherwise specified.

6 Extension of Model-Based approaches to MI Problems

Because of the successful results of both LTGA and iAMaLGaM in their respective domains, a natural question arises: can the model-based abilities of these algorithms be extended to MI problems such that dependences between discrete and continuous vari-

Table 4: Regression coefficients for scalability of evaluations on F_1 through F_4 benchmarks. α_ρ is the regression coefficient where ρ is the fraction of continuous variables in the problem.

	Population Scheme	**Scalability Coefficient**				
		α_0	$\alpha_{0.25}$	$\alpha_{0.5}$	$\alpha_{0.75}$	$\alpha_{1.0}$
F1	Optimal Population-Size	1.5	1.5	1.8	2.0	1.9
	Parameter Free	1.5	1.6	1.8	2.2	1.9
F2	Optimal Population-Size	1.5	1.6	1.8	1.9	1.9
	Parameter Free	1.4	1.7	2.0	2.0	1.9
F3	Optimal Population-Size	1.4	1.7	1.8	1.9	1.9
	Parameter Free	1.7	1.6	1.9	1.9	1.8
F4	Optimal Population-Size	1.6	1.8	2.4	2.2	1.9
	Parameter Free	1.6	1.7	2.5	2.6	2.0

ables are explicitly also considered? We address this question by first considering a few straightforward model-based approaches and by comparing their performance to GAMBIT.

6.1 Algorithms

We consider MI-LTGA and MI-iAMaLGaM: algorithms that extend the individual algorithms from their respective domain to the mixed-integer domain using only discrete or continuous models. Additionally we consider iAMaLGaM+LS where we allow the continuous model-building and sampling of iAMaLGaM to guide the search, while a first-improvement local search hill-climbs in the discrete variable space. Good performance of model-based algorithms, LTGA and iAMaLGaM in their respective domains does not necessarily translate to good performance in the mixed-integer space.

6.1.1 MI-iAMaLGaM

In MI-iAMaLGaM, all binary variables are initialized to either 0 or 1, but further treated as if they were real-valued. These variables are turned back into binary values during function evaluation, where values smaller than 0.5 are considered to be 0, and values greater than 0.5 are considered to be 1.

6.1.2 MI-LTGA

MI-LTGA is the discrete analogy of MI-iAMaLGaM, i.e., all variables are considered discrete.To this end, a discretization of the continuous variables takes place. MI-LTGA uses the well-known Gray encoding. With Gray coding adjacent numbers always differ by exactly one bit. This solves the problem of Hamming cliffs present with decimal encoding, where a small change to the encoded number can cause a completely different representation in the binary space. We furthermore use 24 bits to represent every continuous variable.

6.1.3 iAMaLGaM+LS

Another intuitive approach is to use the domain specific mechanisms for that domain only and perform local search in the other domain. Using local search in one domain effectively eliminates effects of having a variation of partial solutions for that domain, as the local search guarantees finding a local optimum, given that values of the variables from the opposite domain are fixed. In iAMaLGaM+LS all continuous variables of MI problems are modeled and sampled by iAMaLGaM, however each function evaluation is followed by optimization over all discrete variables with first-improvement local search with a bit-flip neighborhood.

6.2 Results

Results are shown on problems with an equal number of discrete and continuous variables. Bisection was used to determine a population size that resulted in at least 29/30 successful runs with the least number of evaluations. The success criterion for a run is reaching a solution whose objective value is within 10^{-10} of the global optimum. Bisection starts by attempting to solve a problem with some small population size (eg. 1, or 10), and if a success criterion is not met the population size is doubled and the process begins anew. Eventually, when a problem is solved for some population size n, a binary search process between n and $n/2$ is used to find the smallest population size for which the problem is still solved. In addition to results obtained via bisection, we also present the performance of GAMBIT with the parameterless scheme.

On the completely independent benchmark F_1 the algorithms perform relatively well, and manage to solve this simple benchmark for all tested problem sizes. iAMaLGaM+LS requires smaller population sizes than other algorithms on F_1 and F_2. However, it performs significantly worse than the other algorithms in terms of function evaluations. MI-iAMaLGaM performs better than iAMaLGaM+LS in terms of function evaluations on F_1 and F_2. However it is not able to consistently solve $F3$ - a problem where strong discrete dependences are present. MI-LTGA can only solve small instances, and becomes very expensive as the problem size increases. GAMBIT is the most consistent and most efficient approach. It is able to solve all problems and performs better than other algorithms in terms of the number of evaluations needed.

The observations from the previous section are reinforced here. The use of the parameterless scheme with GAMBIT produces results which scale with only a constant factor (in a 2.0-3.0 range) difference to the ones of GAMBIT using an optimal population size. Moreover, the parameterless configuration of GAMBIT is still capable of outperforming the alternative approaches.

The results indicate that the tested model-based approaches, regardless of how powerful they can be in their respective domains, are not flexible enough to efficiently extend to the Mixed-Integer domain alone. It is clear from Figure 6 that the interleaved discrete-continuous model-based mechanism of GAMBIT outperforms the single model-based approaches, and warrants further consideration.

7 Explicit Exploitation of Mixed Dependences

One of the key challenges in MI optimization is developing the ability to efficiently identify and exploit dependences which exist across the discrete and continuous domains. So far, GAMBIT addresses any mixed variable dependences only in an indirect way. Some problems that include mixed dependences can be solved efficiently with GAMBIT, because of a well-balanced integrated discrete and continuous model-based mechanism. The goal of this section is to gain insight into the potential impact on the performance of GAMBIT if mixed dependences could be exploited explicitly. To accomplish this, we relax the black-box restriction and consider a setting where the problem structure is known. This allows for the creation of predetermined structures that capture the known problem dependences, without the need to learn them during execution.

One of the key characteristics of GAMBIT is the use of LTGA's linkage tree as

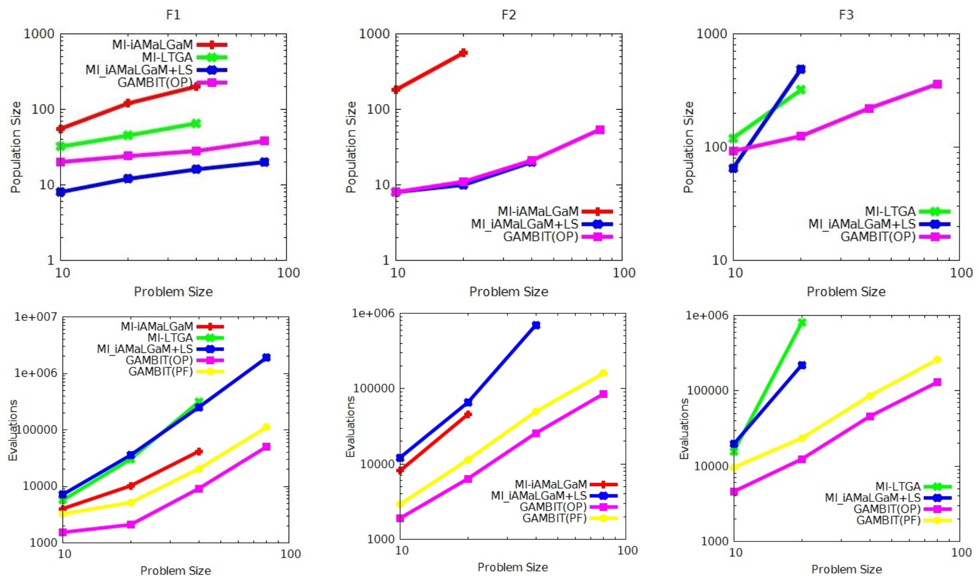

Figure 6: Optimal Population Size and Number of Evaluations for functions F_1, F_2, F_3.

the means to represent discrete variable dependences. Continuous dependences are captured via covariance information following the mechanism of iAMaLGaM. Because the current procedure to generate solutions in GAMBIT is already quite effective, we chose to preserve the current use of the FOS structure to represent the intra-domain dependences, and add mixed dependences on top of it.

7.1 Processing of Mixed Dependences

In this section we introduce a mechanism to GAMBIT which allows for direct processing of mixed dependences, by creating new models for all mixed subsets to sample from. Each such model is built by taking into account only the variables in the subset. This is illustrated in Figure 7. Specifically, each cluster is grouped into smaller clusters using the same mixed distance metric explained earlier in this paper. This results in k equally-sized sub-clusters that describe more closely the relation between configurations of the discrete variables within the mixed subset and those of the continuous variables within the mixed subset. This means that $k^2 * m$ mixed models will be created and maintained throughout the execution of GAMBIT, where m is the number of mixed subsets present in the subset structure. The value for k is set equal to the number of clusters used by the clustering mechanism of GAMBIT detailed in Section 4. This way k does not need to be explicitly specified. Unlike the main continuous model, which is used for sampling the entire continuous space, a sub-model represents only the variables present in those subsets, making it relatively inexpensive in terms of computing resources needed.

In the loop over FOS subsets a new mixed subset type may now be encountered. When a mixed subset is encountered, for each solution a donor solution is randomly selected from the cluster. Depending on the donor, a sub-cluster to which this donor belongs is probabilistically calculated. The discrete variables of the donor are mixed with the solution in question, while the continuous variables are sampled from the model

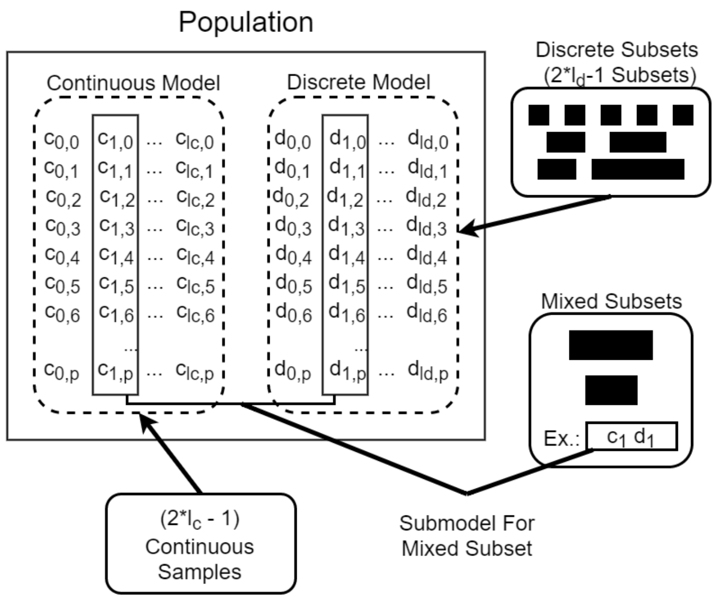
Population
Continuous Model
Discrete Model
$c_{0,0}$ $c_{1,0}$ … $c_{lc,0}$
$c_{0,p}$ $c_{1,p}$ … $c_{lc,p}$
$d_{0,0}$ $d_{1,0}$ … $d_{ld,0}$
$d_{0,p}$ $d_{1,p}$ … $d_{ld,p}$
Discrete Subsets
(2^*l_d-1 Subsets)
Mixed Subsets
Ex.: c_1 d_1
(2^*l_c - 1) Continuous Samples
Submodel For Mixed Subset

Figure 7: Different models are used for sampling different types of subsets.

which was determined to be the match. Changes are immediately evaluated and kept only if it results in an improvement. Otherwise, the change is rejected. The procedure for sampling new solutions when a mixed subset is encountered is summarized in pseudo-code form in Figure 8.

7.2 Performance Results

In this section we consider three ways of handling variable dependences:

- original GAMBIT
- GAMBIT with a predetermined discrete linkage-tree-based FOS
- GAMBIT with a predetermined discrete linkage-tree-based FOS appended with known mixed variable subsets

We compare the three approaches in a parameter-free setting. Functions F_1 and F_4 do not contain any mixed dependences. Because of this lack of mixed dependences, the only comparison possible on these functions is the performance of the original GAMBIT with GAMBIT using a predetermined discrete linkage-tree-based FOS. Since F_1 only consists of independent variables, it may be intuitive that neither configuration outperforms the other, since no exploitable variable dependency structure exists in the problem. More surprisingly, on a problem with a clear underlying structure F_4 also shows no difference in efficiency between the learning and predetermined tree structures. This suggests that GAMBIT is very efficient at learning the structure of this problem.

The structures of functions F_5 and F_6 posses underlying structures which include mixed dependences. The graphs in Figure 9 and the scalability coefficient α in Table 5 indicate improvement in scalability when known mixed dependences are processed.

```
**Sampling a mixed subset for cluster sub-population $\mathcal{P}_k$**
IF SUBSET IS MIXED
    for $i \in \{0, 1, \ldots, k-1\}$ do
        $\mathcal{P}_{k_i} \leftarrow \textsc{SubCluster}(\mathcal{P}_k, Subset)$
        $\mathcal{N}_i \leftarrow \textsc{EstimateRestrictedModel}(\mathcal{P}_{k_i}, Subset_c)$
    $donor \leftarrow \textsc{GetRandomSol}(\mathcal{P})$
    $match \leftarrow \textsc{IdentifyBestModelMatch}(donor_d)$
    for $j \in \{0, 1, \ldots, \mathcal{P}_k.length - 1\}$ do
        $\mathcal{O}_{prev} \leftarrow \mathcal{P}_{k_j}$
        $\mathcal{O}_{j_{sub.d}} \leftarrow \textsc{CopyDiscretePartOfSubset}(donor)$
        $\mathcal{O}_{j_{sub.c}} \leftarrow \textsc{SampleMatchedModel}(\mathcal{N}_{match}, Subset_c)$
        $\mathcal{O}_j \leftarrow \textsc{ReplaceSubsetVariables}(\mathcal{O}_j, \mathcal{O}_{j_{sub.c}}, \mathcal{O}_{j_{sub.d}})$
        $\textsc{EvaluateFitness}(\mathcal{O}_j)$
        if $fitness(\mathcal{O}_j) < fitness(\mathcal{O}_{prev})$ then
            $\mathcal{O}_j \leftarrow \mathcal{O}_{prev}$
    $\mathcal{P}_k \leftarrow \mathcal{O}$
    $return\ \mathcal{P}_k$
```

Figure 8: Pseudo-code for processing and sampling from a mixed subset with GAMBIT for a cluster population $\mathcal{P}_k$.

Table 5: Regression coefficients α for scalability of the differently build FOS structures

Function	Learning Struct.	Predetermined FOS	Predetermined FOS with Mixed Subsets
F_1	1.6	1.5	-
F_4	1.9	1.9	-
F_5	2.5	2.2	1.9
F_6	1.8	1.8	1.5

This is an important observation, as it suggests that if such dependences can be learned dynamically, the improved performance of GAMBIT may carry over into the black-box setting.

7.3 Runtime Results

In the previous section the benefits of using the proposed mixed dependences mechanism with GAMBIT in terms of the number of evaluations needed to find the optimum have been highlighted. This section considers the potential overhead and drawbacks of the new GAMBIT mechanism in terms of execution time. Specifically, we look at the runtime differences of the linkage tree learning approach versus the predetermined tree approach. Additionally we examine the runtime costs of using the mixed subset processing and sampling mechanism, which improved GAMBITs performance in terms of evaluations needed on the benchmark problems. All experiments were conducted on the same machine, with an Intel(R) Core(TM) i7-2760QM CPU dual 2.4GHz processor, 12 GB of RAM, running a 64-bit version of the Windows 10 operating system. Results are averaged over 30 independent runs for each algorithm configuration.

Figure 10 summarizes the performance in runtime. While the results on the independent $F1$ function are very similar in terms of runtime, results on the $F4$ function show that a "Predetermined Tree" configuration is faster. As the performance on $F4$ in terms of the number of evaluations needed is very similar (see Figure 9), the runtime

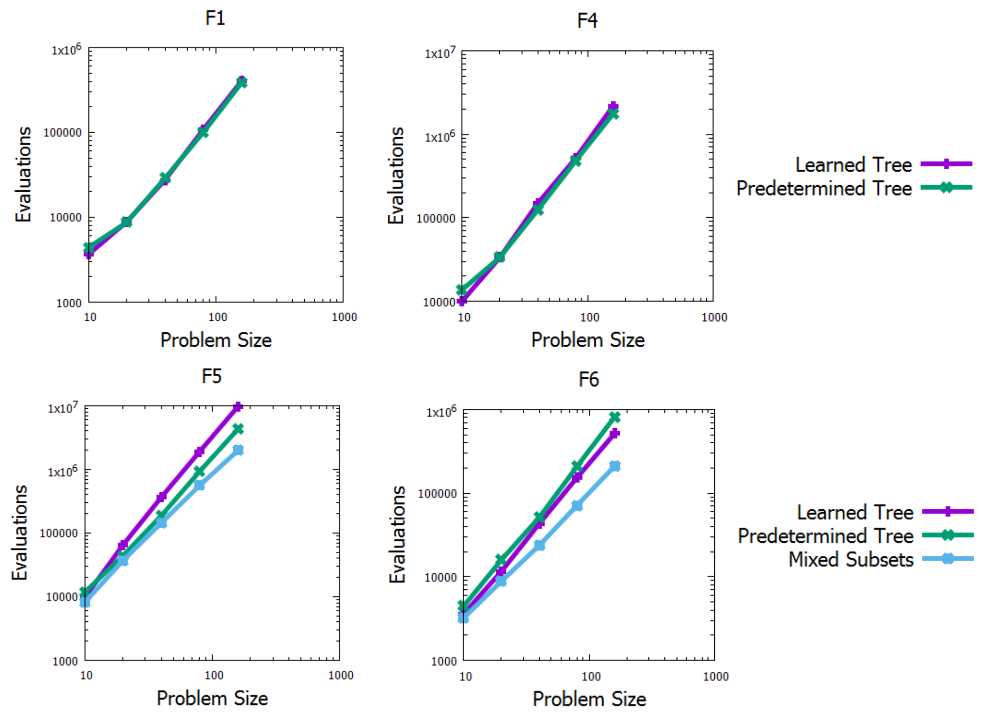

Figure 9: Problem size vs. number of evaluations needed for the differently crafted FOS structures in problems F_1, F_4, F_5 and F_6.

difference can be attributed to the overhead needed to build a new linkage tree with every generation in the "Learned Tree" configuration. This overhead is also a key factor in the $F5$ function.

Results on functions $F5$ and $F6$ show that the performance gain from the inclusion and processing of mixed subsets outweighs the added time cost of the mixed subset mechanism. The "Mixed Subsets" configuration is similar or faster than both the "Predetermined Tree" and "Learned Tree" configurations. This result implies that the ability to competently acquire mixed dependence information can improve optimization performance in terms of both required number of function evaluations and runtime in case of problems where mixed dependencies are present and important.

8 Alternative Approaches

In this section we consider a selection of state-of-the-art algorithms for mixed-integer optimization from research and industry, and examine their ability to solve problems that exhibit different landscape features. Most of commercially available algorithms are known for their ability to efficiently solve problems by exploiting the structure of objective and constraint functions directly, i.e. taking a white-box approach. We wish to examine if these alternative approaches are able to exploit the structure of problems where dependences between problem variables are a consequence of the objective function, or constraints. We test the performance of a selection of Mixed Integer Nonlinear Programming (MINLP) solvers from the GAMS framework. Additionally, we compare our results with the Mixed-Integer Evolution Strategy (MIES) (Li et al. (2013)). Con-

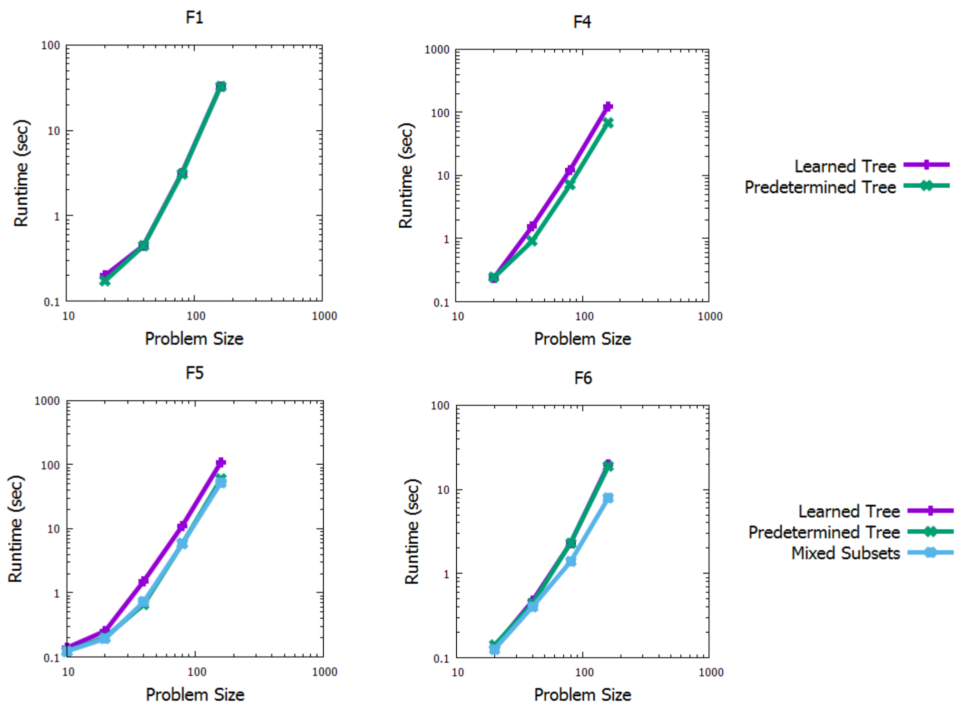

Figure 10: Problem size vs. runtime performance for the differently crafted FOS structures in problems F_1, F_4, F_5 and F_6.

strained Problems (F_6 through F_{12}) vary in problem size, as described in Table 3. The remaining benchmarks were tested on a problem size 20 (largest size for which we could test all the MINLP solvers in the GAMS framework), where half of the problem variables are continuous and the other half are discrete.

8.1 Algorithms

The selection of the algorithms we tested is a part of the General Algebraic Modeling System (GAMS) framework (Bussieck and Meeraus (2004)), which gathers many state-of-the-art modern Mixed-Integer solvers. In this framework, we have tested algorithms that have been designed for MINLP Problems:

- ALPHAECP: a MINLP solver based on the extended cutting plane method (ECP) (Westerlund and Lundqvist (2001)).
- BONMIN: Basic Open-source Nonlinear Mixed Integer programming algorithm by Bonami and Lee (2007), which makes use of branch-and-bound, branch-and-cut, and outer approximation methods.
- COUENNE: Convex Over and Under Envelopes for Nonlinear Estimation algorithm by Belotti (2009), using spatial branch-and-bound techniques.
- DICOPT: DIscrete and Continuous OPTimizer, based on the extensions of the outer-approximation algorithm for the equality relaxation strategy (Grossmann et al. (2002)).

- KNITRO: A software package which makes use of interior-point and active-set methods for solving MINLP problems by Byrd et al. (2006).
- LINDO and LINDOGLOBAL algorithms adapting branch-and-cut methods (Lin and Schrage (2009)).
- LOCALSOLVER: This algorithm is based on a hybrid neighborhood search approach (Benoist et al. (2011)).
- SBB: Algorithm based on the Standard Branch and Bound approach (Bussieck and Drud (2001)).
- SCIP: Solving Constraint Integer Programs by Achterberg (2009), based on the branch-cut-and-price algorithm.

Additionally we compare our results with MIES, which extends the well-known $(\mu+\lambda)$ evolution strategy (ES) for continuous problems to mixed-integer search spaces. MIES generates solutions with the following procedure. The initial population is randomly generated. Two random solutions from the population $P(t)$ act as parent solutions. A recombination operator is applied to the parent solutions, followed by a mutation operator. These operators are defined differently for continuous, nominal discrete and integer problem variables and feature self-adaptive step-sizes and/or mutation probabilities. This procedure repeats until λ offspring solutions are created. The best μ solutions from the union of the μ parent solutions and the λ offspring are carried over into the new population $P(t+1)$ (Li et al. (2013)).

8.2 Results

Table 6 summarizes the results of testing the algorithms mentioned above on all the benchmarks problems. Almost without exception all of the algorithms have no trouble solving any of the constrained problems. The objective function definition of this selection of constrained benchmarks problems is relatively simple. Various equality and inequality constraints contribute to the difficulty of these problems. The algorithms in the GAMS framework are very well equipped to deal with such problems, as they very efficiently exploit many different constraint handling mechanisms. MIES, using the global competitive ranking introduced by Runarsson and Yao (2002) as means of constraint handling, is also capable of solving $F_7 - F_{12}$ as explained in more detail by Li et al. (2013). GAMBIT using a penalty method approach also solves the provided constrained benchmarks.

The performance of the tested algorithms changes however, when the objective function definition becomes more challenging, even in the absence of constraints. Benchmarks $F1$, $F2$ and $F3$ are solved by most algorithms tested. The MIES version provided by Li et al. (2013) does not solve $F_2 - F_5$. MIES's inability to solve F_2 can be remedied by the inclusion of covariance information during the generation of continuous variables. However, MIES cannot capture and exploit the Deceptive Trap function structure, which prevents it from solving $F_3 - F_5$.

When strong dependences are introduced into the problem, the success rate of this set of algorithms goes down drastically. The $F4$ problem, which contains discrete and continuous dependences localized to the respective domains is unsolvable for six out of the eleven tested algorithms. When cross-domain dependences combined with discrete and continuous ones are featured, as in F_5, only two algorithms succeed.

In contrast, GAMBIT is capable of solving all of the problems in this benchmark set. A commercial algorithm ALPHAECP is the only other algorithm tested by us which also solved all the tested problems successfully. The runtimes of both algorithms were comparable. However, commercial algorithms employ a very extensive analysis of the constraint space. GAMBIT does not use complex constraint-handling techniques. Instead, it treats the constrained space as a black-box and utilizes the penalty function method and the clustering mechanism to solve constrained problems.

9 Discussion

It is paramount to reiterate that, by design, GAMBIT aims to explore and exploit problem structure of multi-modal or otherwise challenging objective function landscapes which may contain inter- and intra- variable dependences. Moreover, the design of GAMBIT was done with a BBO setting in mind. The problem types which GAMBIT aims to optimize differ from the problems usually tackled with the MINLP solvers in the GAMS framework. Such solvers are exceptional in the optimization of highly constrained problems, usually with relatively simple objective functions. They explicitly exploit linear constraints and do not treat the constraint space as a black-box. This is demonstrated on problems F_7-F_{12}, where almost all GAMS solvers succeed on every constrained problem. The performance of GAMBIT on these functions is meant to illustrate GAMBITs ability to handle some constrained spaces despite treating the constraints as a black-box. However, highly constrained MINLP problems solvable by the GAMS solvers remains out of reach for GAMBIT. Conversely, the results provided in Table 6 on functions F_1-F_6 show that while GAMBIT succeeds, many state-of-the-art algorithms struggle to solve even low dimensional problems when strong dependences are present in the objective function between the discrete and continuous variables. GAMBIT does not assume anything about the objective function, the constraints, or about their structure. As such GAMBIT targets different types of problems under different assumptions making a direct comparison to GAMS solvers not straightforward. This is not to say that GAMBIT can not be specialized to exploit known problem characteristics such as linear constraints either directly or by hybridization with other solvers, resulting in potentially vastly improved performance in such cases.

10 Conclusions and Future Work

This paper considers the design of model-based EAs for solving Mixed-Integer problems. In order to exploit a problem's structure in a black-box setting, such structure needs to be learned. Model-based EAs have previously proven to very efficient in learning and exploiting various types of structure for discrete and continuous optimization problems, most notably linkage or dependency information. The Genetic Algorithm for Model-Based mixed-Integer opTimization, GAMBIT, that we presented and extended in this article combines the structure learning abilities of LTGA and iAMaLGaM from the discrete and continuous domains respectively for use in the MI domain. In this article we furthermore introduced specific model-building algorithms for the MI domain, and showed that GAMBIT, due to its ability to properly balance discrete and continuous model building and sampling outperforms straightforwardly applied single-model based alternative approaches. Moreover, the proposed approach to exploiting mixed dependencies was shown to improve performance substantially if such dependencies are present and modeled sufficiently accurately. This calls for future work to focus on methods to automatically detect these mixed dependencies accurately and reliably in order to allow GAMBIT to be robust against mixed dependencies even in a BBO setting.

Table 6: Ability of MINLP algorithms to solve benchmarks with different landscape features.

Function:	Unconstrained						Constrained					
Solver	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}
ALPHAECP	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
BONMIN	✓	✓	**no**	**no**	**no**	**no**	✓	✓	✓	✓	✓	✓
COUENNE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	**no**
DICOPT	✓	✓	**no**	**no**	**no**	✓	✓	✓	✓	✓	✓	✓
KNITRO	✓	✓	✓	**no**	**no**	✓	✓	✓	✓	✓	✓	✓
LINDO	✓	✓	✓	✓	**no**	✓	✓	✓	✓	✓	✓	✓
LINDOGLOBAL	✓	✓	✓	✓	**no**	✓	✓	✓	✓	✓	✓	✓
LOCALSOLVER	✓	✓	✓	**no**	**no**	✓	✓	✓	✓	✓	✓	✓
SBB	✓	✓	✓	**no**	**no**	✓	✓	✓	✓	✓	✓	✓
SCIP	✓	✓	✓	✓	**no**	✓	✓	✓	✓	✓	✓	✓
MIES	✓	**no**	**no**	**no**	**no**	✓	✓	✓	✓	✓	✓	✓
GAMBIT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Practicality is one of most important factors in real-world optimization. Many algorithms produce good results only when correctly parameterized, which can be very problematic especially in black-box settings. Determining an optimal, or even a good population size parameter for an EA is not trivial. To address this, a Parameterless scheme was added to GAMBIT. Even though removing the need to specify the population size and number of clusters parameters from GAMBIT creates an evaluation overhead, the scalability of the algorithm is only a small constant factor larger than the one using empirically optimized settings.

The performance of a collection of state-of-the-art MI solvers was compared with GAMBIT. Our results showed that these solvers are very efficient on problems with simpler objective functions, even in the presence of constraints. However, when faced with non-convex objective landscapes, especially ones that contain strong intra-domain or cross-domain dependences, most of the alternative approaches we considered cannot solve even low dimensional cases. GAMBIT performs much more efficiently on these benchmarks. Since real-world problems may exhibit difficult landscapes with complex variable dependences, and may be complex to an extent that a BBO approach is mandatory, our observations suggest that our model-based EA approach has potential to be successful in the MI domain, also for real-world problems.

References

Achterberg, T. (2009). SCIP: Solving Constraint Integer Programs. *Mathematical Programming Computation*, 1(1):1–41.

Belotti, P. (2009). Couenne: a Users Manual. *Technical Report, Lehigh University*.

Benoist, T., Estellon, B., Gardi, F., Megel, R., and Nouioua, K. (2011). Localsolver 1. x: a Black-Box Local-Search Solver for 0-1 Programming. *4OR*, 9(3):299–316.

Bonami, P. and Lee, J. (2007). BONMIN User Manual. 4:2008.

Bosman, P. A. N., Grahl, J., and Thierens, D. (2008). Enhancing the Performance of Maximum-Likelihood Gaussian EDAs Using Anticipated Mean Shift. In *Parallel Problem Solving from Nature – PPSN X*, LNCS, pages 133–143.

Bosman, P. A. N., Grahl, J., and Thierens, D. (2009). AMaLGaM IDEAs in Noiseless Black-Box Optimization Benchmarking. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '09, pages 2247–2254, New York, NY, USA. ACM.

Bosman, P. A. N. and Thierens, D. (2012). Linkage Neighbors, Optimal Mixing and Forced Improvements in Genetic Algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '12, pages 585–592, New York, NY, USA. ACM.

Bussieck, M. and Meeraus, A. (2004). General Algebraic Modeling System (GAMS). In Kallrath, J., editor, *Modeling Languages in Mathematical Optimization*, volume 88 of *Applied Optimization*, pages 137–157. Springer US.

Bussieck, M. and Pruessner, A. (2003). Mixed-integer Nonlinear Programming. *SIAG/OPT Viewsand-News*, 14(1):19–22.

Bussieck, M. R. and Drud, A. (2001). SBB: A New Solver for Mixed Integer Nonlinear Programming. *Talk, OR*.

Byrd, R. H., Nocedal, J., and Waltz, R. A. (2006). KNITRO: An integrated package for nonlinear optimization. In *Large-scale nonlinear optimization*, pages 35–59. Springer.

Emmerich, M., Zhang, A., and Lucas, P. (2008). Mixed-integer Bayesian optimization utilizing a priori knowledge on parameter dependences. In *Proceedings of the 20th BelgiumNetherlands Conference on Articial Intelligence*, BNAIC, pages 65–72.

Grossmann, I. E., Viswanathan, J., Vecchietti, A., Raman, R., Kalvelagen, E., et al. (2002). Gams/DICOPT: A discrete continuous optimization package. *GAMS Corporation Inc*.

Hansen, N. (2011). CMA-ES for Mixed-Integer Nonlinear Optimization. In *Tech. Rep. RR-7751, INRIA*.

Harik, G. R. and Lobo, F. G. (1999). A Parameter-Less Genetic Algorithm. In *IEEE transactions on evolutionary computation*, pages 523–528.

Li, R., Emmerich, M. T. M., Eggermont, J., Bäck, T., Schütz, M., Dijkstra, J., and Reiber, J. H. C. (2013). Mixed Integer Evolution Strategies for Parameter Optimization. *Evolutionary Computation*, 21(1):29–64.

Lin, Y. and Schrage, L. (2009). The global solver in the LINDO API. *Optimization Methods & Software*, 24(4-5):657–668.

Luong, N. H., La Poutré, H., and Bosman, P. A. (2015). Exploiting Linkage Information and Problem-Specific Knowledge in Evolutionary Distribution Network Expansion Planning. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*, GECCO '15, pages 1231–1238, New York, NY, USA. ACM.

Rodrigues, S., Bauer, P., and Bosman, P. A. (2014). A Novel Population-based Multi-objective CMA-ES and the Impact of Different Constraint Handling Techniques. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 991–998, New York, NY, USA. ACM.

Runarsson, T. and Yao, X. (2002). Constrained Evolutionary Optimization. In *Evolutionary Optimization*, volume 48 of *International Series in Operations Research and Management Science*, pages 87–113. Springer US.

Sadowski, K. L., Bosman, P. A., and Thierens, D. (2015). A Clustering-Based Model-Building EA for Optimization Problems with Binary and Real-Valued Variables. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*, GECCO '15, pages 911–918, New York, NY, USA. ACM.

Sadowski, K. L., Bosman, P. A. N., and Thierens, D. (2013). On the Usefulness of Linkage Processing for Solving MAX-SAT. In *Genetic and Evolutionary Computation Conference, GECCO '13, Amsterdam, The Netherlands, July 6-10, 2013*, pages 853–860.

Sadowski, K. L., Thierens, D., and Bosman, P. A. N. (2014). Combining Model-Based EAs for Mixed-Integer Problems. In *Parallel Problem Solving from Nature — PPSN XIII*, LNCS, pages 342–351. Springer International Publishing.

Thierens, D. (2010). The Linkage Tree Genetic Algorithm. In *Parallel Problem Solving from Nature — PPSN XI*, LNCS, pages 264–273, Berlin. Springer–Verlag.

Thierens, D. and Bosman, P. A. N. (2011). Optimal Mixing Evolutionary Algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '11, pages 617–624, New York, NY, USA. ACM.

Westerlund, T. and Lundqvist, K. (2001). *Alpha-ECP, Version 5.01: An interactive MINLP-Solver Based on the Extended Cutting Plane Method*. Åbo Akademi.