EURO J Comput Optim (2013) 1:201–218
DOI 10.1007/s13675-013-0007-y

ORIGINAL PAPER

Restrict-and-relax search for 0-1 mixed-integer programs

Menal Guzelsoy · George Nemhauser · Martin Savelsbergh

Received: 2 September 2012 / Accepted: 21 December 2012 / Published online: 31 January 2013

Abstract A highly desirable characteristic of methods for solving 0-1 mixed-integer programs is that they should be capable of producing high-quality solutions quickly. We introduce restrict-and-relax search, a branch-and-bound algorithm that explores the solution space not only by fixing variables (restricting), but also by freeing, or unfixing, previously fixed variables (relaxing). Starting from a restricted 0-1 mixed-integer program, the branch-and-bound algorithm may, at any node of the search tree, selectively relax, or unfix, previously fixed variables, restrict, or fix, additional variables, or unfix and fix variables at the same time using dual or structural information (problem-specific information). This process yields a dynamic search that is likely to find high-quality feasible solutions more quickly than a traditional search and that is also capable of proving optimality. A proof-of-concept computational study demonstrates its potential. A straightforward generic implementation in SYMPHONY, an open source solver for mixed-integer programs, is shown to generate high-quality solutions for many MIPLIB instances as well as for large-scale multicommodity fixed-charge network flow instances much more quickly than SYMPHONY itself.

Keywords 0-1 Integer programming · Branch-and-bound · Feasible solutions · Restrict-and-relax

Mathematics Subject Classification 90C11 · 65K05

This work was supported by funding from ExxonMobil and the Air Force Office of Scientific Research.

M. Guzelsoy · G. Nemhauser
Georgia Institute of Technology, Atlanta, GA, USA

M. Savelsbergh (✉)
University of Newcastle, Callaghan, Australia
e-mail: martin.savelsbergh@newcastle.edu.au

Introduction

A highly desirable characteristic of methods for solving 0-1 mixed-integer programs (MIPs) is that they should be capable of producing high-quality solutions in the early stages of the computation. From a practical perspective, finding high-quality solutions and doing so quickly is frequently all that is desired. From a methodological perspective, it is important because many of the techniques embedded in integer programming solvers rely on the availability of both lower and upper bounds. Therefore, it is not surprising that substantial effort has been dedicated, especially in the last decade, to enhancing the ability of methods for solving 0-1 MIPs to find high-quality solutions quickly, either by embedding techniques specifically designed for this purpose, or by modifying the methods themselves.

One of the techniques specifically designed to find high-quality solutions and embedded in state-of-the-art commercial branch-and-bound solvers, such as CPLEX, XPRESS, and Gurobi, is relaxation induced neighborhood search (RINS) [2]. RINS explores a part of the search space defined by a known feasible solution and a linear programming solution. Variables with the same value in both solutions are fixed, and the resulting restricted MIP is solved. The hope is that the reduced search space defined by the fixed variables contains high-quality integer solutions and that it can be explored efficiently by solving the restricted MIP. A variant that does not require a known feasible solution is relaxation enforced neighborhood search (RENS) [1].

Local branching [5] is a related, but quite different, example of the effective use of restricted MIPs. In local branching, given a reference solution $\bar{x}$, a neighborhood of the reference solution is defined by those feasible solutions that differ in at most k of the binary variables, i.e., the feasible solutions with a Hamming distance of less than or equal to k from the reference solution. This concept (or more specifically the local branching constraint) is used to define a branching scheme in which in one branch a restricted MIP is solved to find a best solution in the neighborhood, and in the other branch a Hamming distance of greater than or equal to $k + 1$ is enforced. The resulting branch-and-bound algorithm favors improving the incumbent solution, hence producing high-quality solutions in the early stages of the computation.

Yet another example of the effective use of restricted MIPs is branch-and-price guided search (BPGS) [8]. BPGS is based on an extended formulation of the MIP to be solved, in which restrictions of that MIP are enumerated. The extended formulation is solved with a branch-and-price algorithm, which automatically defines and solves restricted MIPs and is guaranteed to produce a provable optimal solution when run to completion. Computational experience demonstrates that the pricing problems quickly produce restricted MIPs that yield high-quality solutions.

The common characteristic of these approaches is that they explore portions of the solution space by defining a restricted MIP and then solving that MIP using a general MIP solver. The idea is that by fixing certain variables to obtain a restricted MIP, the restricted MIP can be solved more quickly and if the variables that were fixed were chosen appropriately, the resulting solution will be a high-quality solution to the original, unrestricted MIP.

In this paper, we introduce restrict-and-relax search, a branch-and-bound algorithm that explores the solution space not only by fixing variables (restricting), but also by freeing, or unfixing, previously fixed variables (relaxing). Starting from a restricted 0-1 mixed-integer program, the branch-and-bound algorithm may, at any node of the search tree, selectively relax, or unfix, previously fixed variables, restrict, or fix, additional variables, or unfix and fix variables at the same time using dual or structural information (problem-specific information). This process yields a dynamic search which tends to find high-quality feasible solutions more quickly than a traditional search. Restrict-and-relax search always works with a restricted, but dynamically changing, set of variables. As a consequence, restrict-and-relax search can be used to tackle instances that are so large that they cannot be fully loaded into memory. Producing high-quality solutions quickly to large MIPs was the main motivation for developing restrict-and-relax search. Another was to investigate whether it is possible to effectively exploit information obtained during branch-and-bound to refine an initial restricted MIP, e.g., either by fixing additional variables or by unfixing previously fixed variables.

It is insightful to highlight the main differences between restrict-and-relax search and some of the other approaches exploiting the benefits of solving restricted MIPs. Restrict-and-relax search is fundamentally different from standard branch-and-bound enhanced with local search to find high-quality solutions in the early stages of the computation (e.g., branch-and-bound enhanced with RINS). The latter approach always works on the full MIP, but, at certain nodes in the search tree, solves a specific restricted MIP. Restrict-and-relax search, instead, always works on a restricted, but changing, MIP, and might only look at the full MIP at the end of the search if it is desired to prove optimality. Restrict-and-relax search never solves a specific restricted MIP. Restrict-and-relax search also differs from traditional IP-based local search, in which some sequence of restricted MIPs is solved, since it can, if desired, yield a proof of optimality, and bounds can be obtained throughout the search.

A computational study demonstrates the potential of restrict-and-relax search. Our generic implementation in SYMPHONY [11], an open source solver for mixed-integer programs, is shown to generate high-quality solutions for many MIPLIB instances as well as for large-scale multicommodity fixed-charge network flow instances much more quickly than SYMPHONY itself.

In the rest of the paper, we first present the core ideas underlying restrict-and-relax search. Then we discuss our proof-of-concept implementation of restrict-and-relax search and we present the results of an extensive computational study. We close with some final remarks and directions for further research.

Restrict-and-relax search

Consider the 0-1 MIP

$$z = \min_{x \in \mathcal{S}} cx, \tag{1}$$

where $\mathcal{S} = \{x \in \mathbb{B}^r \times \mathbb{R}_+^{n-r} | Ax = b\}$. Restrict-and-relax search solves this problem by branch-and-bound, but instead of starting with the full problem, restrict-and-relax search starts with a restricted problem, i.e., a subset of variables indexed by $F \subseteq I =$

$\{1, \ldots, r\}$ is set to one of their bounds, i.e., $x_i = \bar{x}_i$, where $\bar{x}_i$ is either 0 or 1, resulting in the restricted 0-1 MIP

$$z_F = \min_{x \in \mathcal{S}_F} cx, \tag{2}$$

where $\mathcal{S}_F = \{x \in \mathbb{B}^r \times \mathbb{R}_+^{n-r} \mid Ax = b, x_i = \bar{x}_i \; , i \in F\} \subseteq \mathcal{S}$. At each node t of the branch-and-bound tree, the linear programming (LP) relaxation of

$$\begin{aligned} z_t = \min \;\; & cx \\ \text{s.t.} \;\; & Ax = b \\ & x_i = \bar{x}_i \;,\; i \in F \cup B_t \\ & x \in \mathbb{B}^r \times \mathbb{R}_+^{n-r}, \end{aligned} \tag{3}$$

is solved, where $B_t \subseteq I \backslash F$ is the index set of variables fixed by branching decisions. Because the restricted 0-1 MIP given by (3) is a subproblem of the restricted 0-1 MIP given by (2) and thus a subproblem of the original 0-1 MIP given by (1), we have $z_t \geq z_F \geq z$.

The benefit of solving the restricted 0-1 MIP given by (2) is that it is smaller than the original 0-1 MIP and therefore can likely be solved much faster. Unfortunately, there is no guarantee that an optimal, high-quality, or even feasible, solution is found. That depends on the choice of the index set F of variables that is fixed, and the choice of the values to which these variables are fixed. To eliminate this uncertainty, restrict-and-relax search, at any node of the search tree, may unfix previously fixed variables. Furthermore, to maintain the benefit of solving a small, restricted 0-1 MIP, restrict-and-relax search, at any node of the search tree, may also fix additional variables.

That is, at certain nodes of the search tree, the sets of fixed and unfixed variables are altered, i.e., at some node t, we replace F with $\bar{F}$ and explore the part of the search space defined by

$$\begin{aligned} \bar{z}_t = \min \;\; & cx \\ \text{s.t.} \;\; & Ax = b \\ & x_i = \bar{x}_i \;,\; i \in \bar{F} \cup B_t \\ & x \in \mathbb{B}^r \times \mathbb{R}_+^{n-r}. \end{aligned} \tag{4}$$

The goal, and the challenge, is to choose $\bar{F}$ in such a way that $\bar{z}_t < z_t$ and $\bar{F}$ does not differ much from F. We use dual information associated with the solution to the LP relaxation to guide the choice of variables to fix and to unfix. It is advantageous to change F gradually and solve a sequence of LP relaxations

$$\begin{aligned} \min \;\; & cx \\ \text{s.t.} \;\; & Ax = b \\ & x_i = \bar{x}_i \;,\; i \in F_t^j \cup B_t \\ & x \in \mathbb{R}_+^n, \end{aligned} \tag{5}$$

for $j = 1, \ldots, k$, where $F_t^0 = F$ and $|F_t^j \Delta F_t^{j-1}|$ is small, i.e., the number of variables in the symmetric difference of two consecutive sets of fixed variables is small. This not only guarantees that the linear programs can be solved quickly, it also ensures that up-to-date dual information is available (and used) to guide the choice of variables to fix and to unfix in each iteration. We store any integer-feasible solutions found along the way and then either prune the node or branch on a variable x_i, $i \in I \setminus \{F_t^k \cup B_t\}$.

Observe that the choice of variables to unfix (and fix) is based on local information, i.e., information that is relevant to the restricted 0-1 MIP defined by the node in the branch-and-bound tree. The hope and expectation is that this quickly guides the search into promising parts of the solution space; therefore, producing high-quality solutions early in the search (and thus avoids exploring unnecessary parts of the solution space). Note too that restricting and relaxing serve two different purposes in restrict-and-relax search. Restricting serves to increase the efficiency by reducing the size of the linear programs that need to be solved, whereas relaxing serves to increase the solution quality by expanding the space that is explored.

Recognizing that in a traditional branch-and-bound tree, the levels corresponds to the number of variables fixed by branching, restrict-and-relax search can be interpreted as starting at a node at level k, where k is the number of variables fixed initially, and then jumping up and down the tree as variables are unfixed and fixed. The rules used for pruning nodes are the same as in traditional branch-and-bound search, unless restrict-and-relax search is turned into a heuristic by pruning some nodes without proof that they cannot contain an optimal solution (see below for more details).

The key to success for restrict-and-relax search is choosing the variables to fix and to unfix effectively. It is natural to use the primal and dual information available at a node to do so. This reflects one of the main thrusts underlying restrict-and-relax search, namely exploiting local information to guide the search for high-quality integer solutions.

We fix variables only at LP-feasible nodes and choose variables to fix that are likely to preserve feasibility and that are not likely to cause an increase in LP value when other variables are fixed. More precisely, we choose variables to fix (at LP-feasible nodes) from among the set

$$\{x_i \mid i \in I \backslash F_t \cup B_t \quad \text{and} \quad x_i^* = 0 \quad \text{or} \quad x_i^* = 1\},$$

that is, the set of unfixed integer variables that are set either to their lower or to their upper bound in the current LP solution x^*. For an unfixed variable x_i, LP duality for an optimal primal–dual solution pair yields

$$\text{if } x_i^* = 0, \text{ then } r_i^* \geq 0 \quad \text{and} \quad \text{if } x_i^* = 1, \text{ then } r_i^* \leq 0,$$

where r_i^* is the reduced cost of variable x_i. We choose to fix variables in non-increasing order of the absolute value of their reduced costs. Thus, we give priority to variables with large slacks in their corresponding dual constraints, so that when the dual solution is no longer optimal for a subsequent subproblem, the contribution of these constraints to the violation of optimality is likely to be minimal.

Unfixing variables at a node is more involved. There are three cases to consider: (1) the LP is feasible and the LP value is less than the value of the best known feasible solution; (2) the LP is feasible and the LP value is greater than or equal to the value of the best known feasible solution; and (3) the LP is infeasible. In case (1), we choose variables to unfix that are likely to cause a decrease in the LP value. For a fixed variable $x_i, i \in F_t$, we have that if $x_i^* = 0$ and $r_i^* < 0$ or if $x_i^* = 1$ and $r_i^* > 0$, then unfixing this variable may result in the current solution no longer being optimal and thus yielding a new problem with a strictly lower LP solution value. Hence, we only unfix variables with this property and, similar to fixing variables, we choose to unfix variables in non-increasing order of the absolute value of their reduced costs. In case (2), we cannot fathom the node as normally would be done in a branch-and-bound algorithm, since it is possible that by unfixing some of the fixed variables, the LP value will become less than the value of the best known feasible solution. Therefore, in this situation, we unfix variables as described above. However, care has to be taken with the implementation. Within a branch-and-bound tree, MIP solvers use the dual simplex algorithm to solve LPs and stop as soon as the LP value exceeds the value of the best known feasible solution, and therefore can terminate without producing a primal LP solution. In fact, there may not even be a feasible primal solution. Therefore, if there is a cut-off, we remove it and resolve the LP to obtain its true status. If it is feasible, we proceed as before, i.e., we use dual information to unfix variables. Case (3) is similar to case (2), but the dual is now unbounded. In this case, we have found that more drastic action is needed to see if we can fathom the node. If the number of fixed variables is not too big, we unfix all previously fixed variables (i.e., all but the variables fixed by branching) and resolve the LP. If the LP is feasible and has an objective value below the value of the best known feasible solution, we unfix the variables that have a value different from their previously fixed value. For instance, if a variable was fixed to 1, but now has a value 0.6, then it will be unfixed. If the LP is feasible but has an objective value above the value of the best known feasible solution, or the LP is infeasible, the node is discarded, since it has been completely processed.

Observe that the scheme for handling nodes with an infeasible LP ensures that only nodes that can truly be discarded are discarded and thus the proposed method is capable of solving the instance to proven optimality. However, the scheme involves solving the full LP, which we may not want to (or may not be able to) do for very large instances. In these situations, restrict-and-relax search can be converted into a heuristic by either accepting the LP infeasibility immediately, or by exploring what happens when a small number of previously fixed variables is unfixed.

The success of restrict-and-relax search may depend on the quality of the initial restriction. Three natural choices for initial fixings are: (1) choose the variables to fix based on some known feasible solution; (2) choose the variables to fix based on the solution to the linear programming relaxation; or (3) choose the variables to fix based on the initial feasible solution to the linear programming relaxation, i.e., the solution after Phase I. The last option is of interest when dealing with extremely large instances or when dealing with instances for which it is very time-consuming to solve the linear programming relaxation to optimality.

We have employed strategies (2) and (3). That is, we fix variables with integral values in an LP solution. More specifically, if $x_i^{\mathrm{LP}} = 0$, then we assign a score $s_i = -c_i$, and if $x_i^{\text{LP}} = 1$ then we assign a score $s_i = c_i$. We then fix variables in nondecreasing order of these scores, so that we first fix variables that, when relaxed, increase the value of LP solution more than others. Note that this implies that fixing is random for variables with the same objective coefficients. We fix at most 90 % of the binary variables.

The use of cutting planes in modern integer programming solvers has contributed substantially to their success. Cutting planes are generated and added at selected nodes of the branch-and-bound tree to obtain a tighter polyhedral approximation of the convex hull of the integer feasible set.

The use of cutting planes in restrict-and-relax search requires careful consideration because inequalities generated at a node t for the subproblem defined by F_t^j, i.e., the subproblem given by (5), may not be valid for the subproblem defined by F_t^{j+1} (unless $F_t^{j+1} \subseteq F_t^j$) and for the subproblem associated with a node s in the subtree rooted at t (unless $F_s \subseteq F_t^j$). The valid inequalities also might cut off feasible solutions when variables are relaxed.

To be able to use cutting planes in restrict-and-relax search, we have a few options:

- All existing cutting planes are removed as soon as variables are relaxed. (Of course new cutting planes can be generated for the new subproblem.)
- All relaxed variables are lifted in all existing cutting planes using some lifting algorithm.
- Cutting planes are generated in such a way that they are globally valid, where globally valid is with respect to $\mathcal{S}$ in problem (1), i.e., the cutting planes are generated assuming that all variables (other than variables fixed by branching) are relaxed.

In our current implementation we have adopted the last option, but the second option could produce much tighter relaxations.

Implementation

We have presented a generic version of restrict-and-relax search and a variety of implementations is possible. We have implemented a version controlled by a large number of parameters to allow us to experiment with different setups. The control parameters are listed and defined in Table 1.

Let t be a node with a tree level that is multiple of lf, that is greater than $d^{\min}$, and that is less than $d^{\max}$, or a node that is to be pruned by bound (if pb is enabled), or a node that is to be pruned by infeasibility (if pi is enabled). Then, we fix and relax at node t at least once and at most tl times until the node LP relaxation is feasible and has a solution value strictly less than the current upper bound.

Varying the parameter values can have significant impact on the performance on specific instances. The default values provide a compromise that seems to perform reasonably well across the instances in our test set.

The integer programs associated with child nodes in a traditional branch-and-bound tree always correspond to restricted versions of the integer program associated with the parent node. Consequently, the LP value at the parent node is less than or equal

Table 1 Restrict-and-relax search parameters

Min-depth ($d^{\min}$)	Fix and unfix only at nodes below tree level $d^{\min}$ (default: 0)
Max-depth ($d^{\max}$)	Fix and unfix only at nodes above tree level $d^{\max}$ (default: 50)
Level-frequency (lf)	Fix and unfix at a node if the node level is a multiple of lf (default: 3)
Trial-limit (tl)	At a node, fix and unfix at most tl times (default: 5)
Unfix-ratio (ur)	In a trial, unfix at most ur % of the fixed variables (default: 5)
Fix-ratio (fr)	In a trial, fix at most fr % of the unfixed variables (default: 5)
Prune-by-bound (pb)	If enabled, fix and unfix at nodes pruned by bound regardless of the node level (default: enabled)
Prune-by-infeasibility (pi)	If enabled, fix and unfix at nodes pruned by infeasibility regardless of the node level (default: enabled)

to the LP value at the child nodes. It is because of this property that a node (and the subtree rooted at the node) can be pruned when the LP value at the node is greater than or equal to the value of the best known feasible solution. In restrict-and-relax search, the property that the integer programs associated with child nodes in the search tree always correspond to restricted versions of the integer program associated with the parent node is no longer true, since fixed variables in the integer program associated with the child node may be relaxed. Therefore, a node is not pruned unless one of the following conditions occurs:

- all integer variables (other than the variables fixed by branching) are unfixed and the node LP yields an integer feasible solution;
- all integer variables (other than the variables fixed by branching) are unfixed and the node LP is infeasible or the node LP value is greater than the value of the best known feasible solution; or
- fixing and unfixing has been performed tl times and the node LP remains either infeasible or has a value that is greater than the value of the best known feasible solution.

Because of the trial limit tl, our current implementation does not prove optimality of the solution produced, even if the search tree is completely explored, since it is possible that nodes containing an optimal solution are discarded. This is reasonable since our goal is to achieve high-quality solutions, and the approach is not necessarily intended to be run until the search tree is completely explored.

We have embedded restrict-and-relax search in the development version of SYMPHONY [11] with CLP (the COIN LP solver) because SYMPHONY provides the capability to start from a restricted problem and to fix and unfix variables at selected nodes in the search tree (functionality that is not available in the commercial solvers CPLEX, Gurobi, and XPRESS).

SYMPHONY itself contains various techniques aimed at finding feasible solutions quickly, e.g., two rounding heuristics, six diving heuristics, an implementation of the feasibility pump [4], local branching and RINS, and restrict-and-relax search benefits from (and to some extend relies on) their effectiveness.

Computational study

We have conducted a computational study to assess the potential of restrict-and-relax search. The goal of the computational study is to compare traditional branch-and-bound with restrict-and-relax search with regard to finding good solutions quickly. The computational study has two parts. In the first part, we use restrict-and-relax search to quickly find high-quality solutions to 0-1 MIPs from the well-known test set MIPLIB [10]. In the second part, we use restrict-and-relax search to quickly find high-quality solutions to instances of the multicommodity fixed-charge network flow problem. All experiments were run on an Intel Xeon E5520 processor at 2.27 GHz.

We use restrict-and-relax search with the default settings for the control parameters to quickly find high-quality solutions for a subset of 0-1 MIPs from the well-known test set MIPLIB 2010 [10]. The subset of instances used in our computational experiments was obtained by discarding the following instances from MIPLIB, where we report the number of instances discarded in parentheses:

- Instances with general integer variables (72);
- Instances which are infeasible, the instances labeled I in MIPLIB (15);
- Instances which are extra large, the instances labeled X in MIPLIB (9);
- Instances which are unstable, the instances labeled U in MIPLIB (16);
- Instances which are unstable for either CLP or SYMPHONY (23);
- Instances which are easy, where easy is defined as being solved to optimality by SYMPHONY in less than 500 s (34).

Additionally, we discard instances in which fewer than 60 % of the 0-1 variables are at one of their bounds in the solution to the LP relaxation. The implementation of restrict-and-relax search used in this proof-of-concept study only incorporates a simple scheme for creating the initial restriction. Because restrict-and-relax search is likely to be more successful when the initial restricted integer program is significantly smaller than the original integer program, we limit ourselves to instances where the simple scheme for creating the initial restriction does produce significantly smaller restrictions. This highlights the need for further research on more sophisticated methods for creating the initial restricted integer program. An additional 65 instances were discarded, leaving us with a test set containing 127 instances.

To assess the potential of restrict-and-relax search, we compare the best solution obtained for the instances in our test set with four different approaches:

1. Running the default solver (SYMPHONY) on the full MIP (denoted by Def-full);
2. Running the default solver (SYMPHONY) on the initial restricted MIP (denoted by Def-restricted);
3. Running restrict-and-relax search in which the fixing of variables during the search is disabled (denoted by RR-relax-only);
4. Running restrict-and-relax search (denoted by RR).

The reason for including the variant of restrict-and-relax search in which the fixing of variables during the search is disabled is to investigate the importance of relaxing, which is the fundamental idea of restrict-and-relax search. Since the primary goal of restrict-and-relax search is to quickly find high-quality solutions, we investigate a range of time limits: 100, 250, 500, 1,000 and 2,000 s.

Table 2 Performance statistics for RR-relax-only

	Def-full					Def-restricted				
	100 s	250 s	500 s	1,000 s	2,000 s	100 s	250 s	500 s	1,000 s	2,000 s
Feas	8	8	12	4	4	14	21	26	25	27
<	35	48	53	56	51	39	45	49	50	50
=	11	11	13	14	20	7	3	3	4	4
>	26	20	17	21	23	20	18	17	16	17
No-feas	4	4	3	4	4	0	2	1	1	0

Table 3 Performance statistics for RR

	Def-full					Def-restricted				
	100 s	250 s	500 s	1,000 s	2,000 s	100 s	250 s	500 s	1,000 s	2,000 s
Feas	7	7	10	3	3	13	21	24	24	26
<	34	46	49	52	47	42	43	52	52	54
=	14	13	14	17	20	6	3	3	4	2
>	24	21	21	23	27	18	20	15	15	15
No-feas	4	3	2	3	4	0	2	0	0	0

In Tables 2 and 3, we show for how many instances restrict-and-relax search produced a feasible solution where the default solver did not (feas), for how many instances restrict-and-relax search produced a better solution (<), an equally good solution (=), and a worse solution (>), and for how many instances restrict-and-relax search produced no feasible solution where the default solver did (no-feas). The counts are taken over those instances for which at least one of the approaches produced a feasible solution, which is the reason why the sum of the counts is not the same for each column.

We observe that both variants of restrict-and-relax search perform significantly better than the default solver when either given the full MIP or the initial restricted MIP. Furthermore, the benefits of restrict-and-relax search are most clearly seen with a time limit of 500 s. When the time limit is smaller, a relatively large portion of the time is consumed by root node processing and the benefits of the dynamic exploration of the search space cannot be fully realized. When the time limit is larger, the benefits of the dynamic exploration of the search space, i.e., finding a high-quality solution quickly, is less significant, as the default solver has sufficient time to find a high-quality solution as well. The ability of restrict-and-relax search to quickly find feasible solutions can be observed for the smaller time limits, since the number of instances where restrict-and-relax search finds a feasible solution and the default solver does not is higher than the number of instances where the default solver finds a feasible solution and the restrict-and-relax search does not.

A direct comparison between RR-relax-only and RR, given in Table 4, shows that RR has a slight edge when it comes to finding feasible solutions quickly, whereas RR-relax-only has a slight edge when it comes to finding high-quality solutions. For these

Table 4 Performance statistics for RR

	RR-relax-only				
	100 s	250 s	500 s	1,000 s	2,000 s
Feas	1	1	2	1	2
$<$	19	21	25	26	26
$=$	39	41	40	40	41
$>$	21	24	28	28	29
No-feas	0	1	1	1	1

instances, the efficiency gains from fixing additional variables, do not seem to have an impact. (The situation is different for the multicommodity fixed-charge network flow instances used in the computational experiments.)

The counts presented and discussed above give an initial indication of the potential of restrict-and-relax search, but, at the same time, provide only a limited view. Therefore, we next provide a more comprehensive comparison of the four approaches by means of performance profiles [3]. Let γ_i^a denote a performance metric and $r_i^a = \frac{\gamma_i^a}{\min_a \gamma_i^a}$ denote the relative performance of solution approach a on instance i, respectively, where the value of the performance metric is nonnegative and a smaller value is preferred. The function $\rho^a(\tau) = \frac{|i \in I : r_i^a \leq \tau|}{|I|}$ gives the fraction of instances for which solution approach a is within a factor τ of the best. The performance profile for solution approach a is the graph of $\rho^a(\tau)$ on a log scale. In general, the higher the graph of a solution approach, the better its relative performance. The performance metric used to compare Def-full, Def-restricted, RR-relax-only, and RR is the gap between the value of the best feasible solution found by the approach and the value of the best known feasible solution, where for most instances the value of the best known feasible solution is taken directly from the information provided in the MIPLIB distribution, and where for those instances for which there is no information regarding the value of the best known solution in the MIPLIB distribution, the value of the best feasible solution produced by commercial solver CPLEX (version 12.4) when given 10 h of computing time. The performance profiles for the different time limits are shown in Fig. 1, where the performance profile for a given time limit is computed over the set of instances where at least one of the approaches found a feasible solution.

The performance profiles highlight different aspects of the performance of the solution approaches. When $\tau = 1$, the performance profile shows the fraction of instances for which a solution approach performs best. We see that when the time limit is 100 s, the fraction of instances for which a solution approach performs best is similar for RR, RR-relax-only, and Def-full, a little over 40 %. As we observed before, when the time limit is only 100 s, the benefits of restrict-and-relax search cannot always be realized because a relatively large fraction of the time is spent on processing the root node. When the time limit is between 250 and 1,000 s, we see that RR and RR-relax-only do substantially better than Def-full, performing best in about 50 % of the instances compared to about 30 % for Def-full. The difference in performance between RR and RR-relax-only and Def-full is a bit less when the time limit is 2,000 s, but still

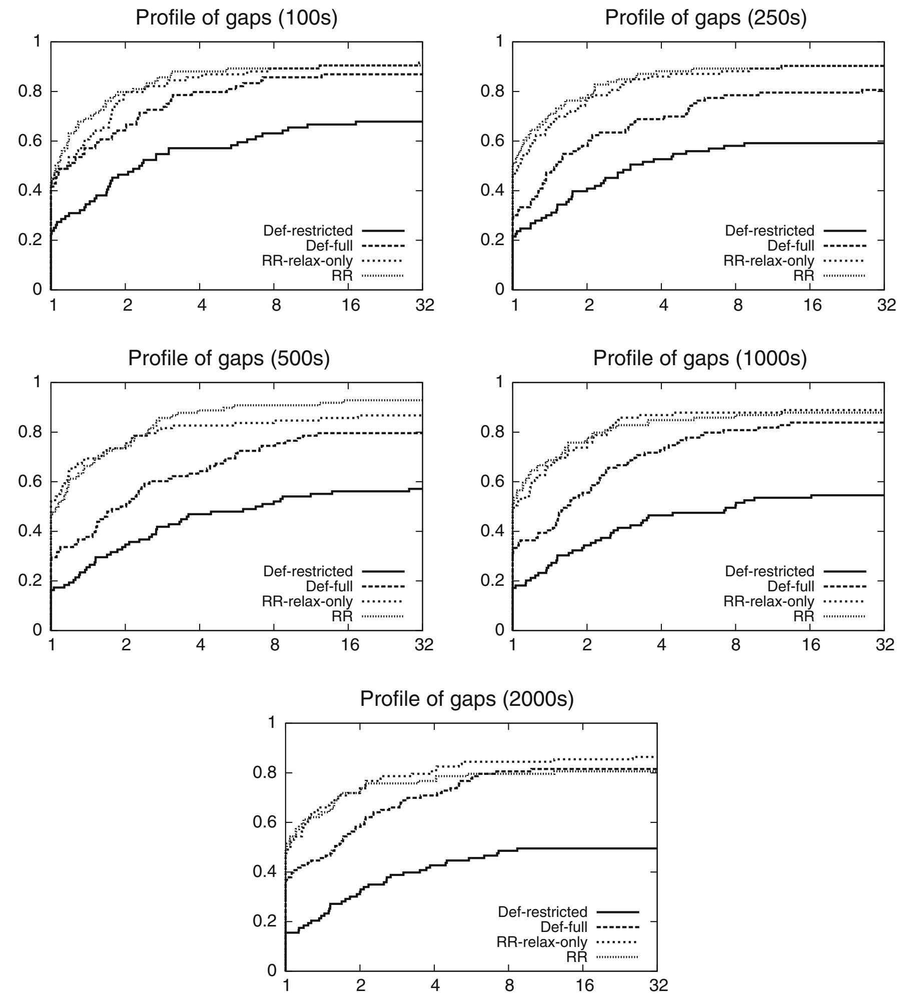

Fig. 1 Performance profiles for MIPLIB instances

substantial. It is important to realize that when given enough time, Def-full should be at least as good as RR and RR-relax-only because it will find an optimal solution (and possibly even better because our implementation of RR and RR-relax-only does not guarantee that they find an optimal solution). However, apparently the time limit of 2,000 s is not enough for Def-full to catch up. When $\tau = 32$, the performance profile essentially shows the fraction of instances for which a solution approach has found a feasible solution (out of the set of instances for which at least one of the solution approaches has found a feasible solution). We see that when the time limit is either 250 or 500 s RR and RR-relax-only do noticeably better than Def-full, whereas for the other time limits the difference is minor, although it exists. This clearly demonstrates the ability of restrict-and-relax search to quickly find high-quality feasible solutions;

if only limited time is available to find a feasible solution, restrict-and-relax search should be the method of choice. When τ is between 2 and 4, we get an indication of the average performance of the solution approaches. A high value indicates that a solution approach may not necessarily have been the best, but it has not been far from the best. The fact that we observe a relatively large difference between RR and RR-relax-only and Def-full in this range, again especially when the time limit is 250 or 500 s, shows the robustness of restrict-and-relax search. Overall, the performance profiles demonstrate decidedly that restrict-and-relax search has the ability to quickly find high-quality solutions. The performance profiles also demonstrate indisputably that relaxing is crucial to the success of restrict-and-relax search since the performance of Def-restrict, which solves the initial restricted integer program, is far worse than the other three approaches.

The interpretation of the performance profiles is somewhat complicated by the fact that we are comparing four approaches and for a particular instance the best solution can be found by any one of the four approaches. The interpretation is more straightforward when we compare two approaches. Therefore, in Fig. 2, we show the performance profiles of just Def-full and RR. The benefits of restrict-and-relaxed search are even more obvious.

Finally, in Tables 5 and 6, we report on the quality of the solutions produced by restrict-and-relax search. More specifically, we show for various gap ranges the number of instances for which restrict-and-relax search produced a better solution ($<$) than the default solver, an equally good solution ($=$), and a worse solution ($>$) within 500 s, e.g., for 25 instances RR produced a solution with a gap of between 1 and 5 % with 18 of them better than, 1 of them equally good, and 6 of them worse than the result of Def-full.

We see that the quality of many of the solutions produced by RR is quite good, but that for some instances the quality of the solution is still quite poor. It is important to remember that some of the instances in the test set are quite large and that only relatively few nodes have been evaluated in the 500 s.

In the second part of our computational study, we investigate whether the generic restrict-and-relax search implementation is able to find high-quality solutions to a set of large-scale instances of integer multicommodity fixed-charge network flow problems (see Hewitt et al. [7] for a description of the problem and the instances). The size of these instances varies between 150,000–600,000 variables, 180,000–700,000 constraints and 750,000–3,000,000 nonzero elements.

Even solving the LP relaxation for some of these instances in a reasonable amount of time can be challenging. Therefore, we construct the initial restricted IP using the primal feasible LP solution obtained by the Phase I simplex algorithm. The LP relaxation of the restricted IP is then solved to obtain the dual information used for fixing and unfixing at the root of the search tree.

Recall that when an infeasible LP is encountered, the generic implementation of restrict-and-relax search unfixes all previously fixed variables (i.e., all but the variables fixed by branching) and resolves the LP. Because we do not want to solve the full LP, when an infeasible LP is encountered during the search, unfixing is done using dual information associated with the LP solution at the parent node. If the resulting LP is feasible and has an objective value below the value of the best known feasible solution,

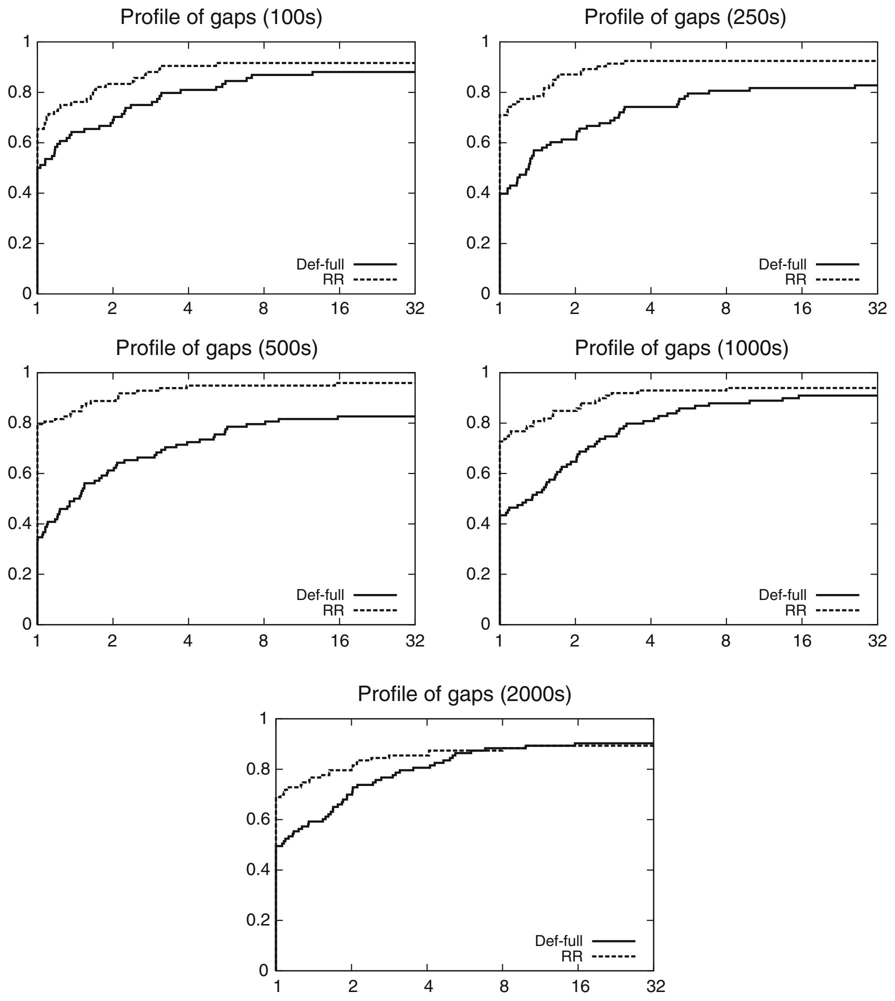

Fig. 2 Performance profiles for MIPLIB instances

then variables that have a value different from their previously fixed value are unfixed (otherwise the node is fathomed).

Finally, based on some preliminary experimentation, the following parameter values were used: max-depth, 1,000; trial-limit, 100; unfix-ratio, 0.50 %; and fix-ratio, 1.0 %.

We report results for time limits of 15 and 60 min in Tables 7 and 8. The instances are identified using the following notation: *T*-*x*-*y*-*z*, where *x* denotes the number of nodes divided by 100, *y* denotes the number of arcs divided by 1,000, and *z* denotes the number of commodities. We also include the value of the LP relaxation at the root (or $-\infty$ when no LP solution has been found). For the four approaches, i.e., Def-full, Def-restricted, RR-relax-only, and RR, we report the value of the best IP solution found and the number of IP solutions found. For each of the instances, the value of the

Table 5 Optimality gaps for solutions produced by RR-relax-only in 500 s

	Def-full			Def-restricted		
	$<$	$=$	$>$	$<$	$=$	$>$
[0, 1)	11	9	1	18	2	1
[1, 5)	15	1	6	16	0	6
[5, 10)	10	2	1	12	0	1
[10, 20)	10	0	5	13	0	2
[20, 50)	8	1	1	9	1	0
[50, 100)	3	0	3	6	0	0
[100, ∞)	8	0	0	1	0	7

Table 6 Optimality gaps for solutions produced by RR in 500 s

	Def-full			Def-restricted		
	$<$	$=$	$>$	$<$	$=$	$>$
[0, 1)	11	11	1	21	1	1
[1, 5)	18	1	6	19	0	6
[5, 10)	8	1	2	11	0	0
[10, 20)	6	0	4	10	0	0
[20, 50)	6	1	4	8	2	1
[50, 100)	2	0	3	5	0	0
[100, ∞)	8	0	1	2	0	7

best IP solution found is shown in bold type face. We note that for all of the instances, 90 % of the variables were fixed in the creation of the initial restricted IP.

A number of observations can be made regarding these results. First, we see that solving the LP relaxation of these instances is indeed quite challenging. Even with a time limit of 60 min, CLP is unable to solve the LP relaxation for 6 out of the 11 instances. (With CPLEX version 12.4 similar results are obtained; CPLEX is unable to solve the LP relaxation within 60 min for 7 out of the 11 instances.) Secondly, we observe that restrict-and-relax search is able to produce IP solutions for all of the instances within 15 min. In fact, for all of the instances, restrict-and-relax search has generated many feasible IP solutions within 15 min. This clearly indicates the enormous potential of restrict-and-relax search for extremely large instances of certain classes of integer programs. Thirdly, we observe that for these instances, there is a clearly observable difference between the performance of RR and RR-relax-only. A closer examination of the results shows that RR evaluates far more nodes than RR-relax-only (75 % more with a time limit of 15 min and 150 % more with a time limit of 60 min), which is almost certainly due to the fact that large numbers of variables are fixed at nodes in the tree resulting in smaller linear programs being solved at nodes. RR therefore explores a much larger portion of the solution space than RR-relax-only.

Table 7 Results for multicommodity fixed-charge network flow instances with a 15-min time limit

	z_{LP}	Def-full		Def-restricted		RR-relax-only		RR	
		z	# sol found	z	# sol found	z	# sol found	z	# sol found
T-5-2-150	$-\infty$	∞	0	16,791,739	1	10,633,431	12	**6,348,474**	110
T-5-2-200	$-\infty$	∞	0	18,264,978	1	10,295,957	13	**6,971,284**	106
T-5-2.5-100	3,010,123	**3,096,873**	1	11,421,658	1	6,680,455	13	3,816,954	62
T-5-2.5-150	$-\infty$	∞	0	16,519,891	1	12,725,369	5	**7,778,874**	101
T-5-2.5-200	$-\infty$	∞	0	22,803,549	1	19,499,182	4	**16,915,379**	22
T-5-3-100	$-\infty$	∞	0	15,336,883	1	10,976,480	4	**6,990,546**	111
T-5-3-125	$-\infty$	∞	0	13,856,412	1	8,983,917	6	**5,822,540**	59
T-5-3-150	$-\infty$	∞	0	15,464,987	1	12,539,533	4	**8,173,244**	31
T-5-3-200	$-\infty$	∞	0	23,302,848	1	20,855,570	2	**14,119,195**	50
T-5-3-50	2,143,902	**2,310,557**	1	7,539,845	3	3,818,608	15	2,858,502	64
T-5-3-75	$-\infty$	∞	0	11,478,392	1	6,484,868	17	**4,622,346**	49

Table 8 Results for multicommodity fixed-charge network flow instances with a 60-min time limit

	z_{LP}	Def-full		Def-restricted		RR-relax-only		RR	
		z	# sol found	z	# sol found	z	# sol found	z	# sol found
T-5-2-150	5,080,051	7,455,350	1	16,791,739	1	9,666,279	19	**6,048,452**	133
T-5-2-200	5,410,777	∞	0	18,264,978	1	9,603,211	16	**6,659,686**	123
T-5-2.5-100	3,010,123	**3,096,873**	1	11,421,658	1	5,440,807	27	3,459,553	75
T-5-2.5-150	$-\infty$	∞	0	16,519,891	1	11,867,108	9	**6,963,319**	146
T-5-2.5-200	$-\infty$	∞	0	22,803,549	1	18,937,251	5	**12,827,664**	83
T-5-3-100	$-\infty$	∞	0	15,336,883	1	10,967,825	5	**6,797,231**	120
T-5-3-125	$-\infty$	∞	0	13,856,412	1	8,715,813	7	**5,115,193**	99
T-5-3-150	$-\infty$	∞	0	15,464,987	1	11,365,537	5	**6,752,942**	88
T-5-3-200	$-\infty$	∞	0	23,302,848	1	19,307,474	4	**10,569,513**	144
T-5-3-50	2,143,902	**2,310,557**	1	7,539,845	3	3,055,353	27	2,858,502	64
T-5-3-75	3,540,911	∞	0	11,478,392	1	6,293,908	23	**4,545,794**	53

Final remarks

The development of restrict-and-relax search was motivated by the need to quickly find high-quality solutions to very large integer programs. However, as our computational study demonstrates, restrict-and-relax search has significant benefits also when solving much smaller integer programs. Our proof-of-concept implementation of restrict-and-relax search in SYMPHONY is effective in producing high-quality solutions to many of the integer programs in MIPLIB and to large-scale multicommodity fixed-charge network flow instances much more quickly than SYMPHONY itself. Perhaps even more important, it challenges the paradigm that has been at the heart of branch-and-bound algorithms since their inception: rather than organizing the search around restrictions only, restrict-and-relax search organizes the search around restrictions and relaxations. It has long been recognized that the basic branch-and-bound paradigm has weaknesses (e.g., that it is difficult to recover from "unfortunate" branching decisions at the top of the tree) and dynamic strategies involving restarts have been proposed to address these weaknesses. Restrict-and-relax search provides a fundamentally different perspective on how to explore the space of feasible solutions.

As mentioned above, we have used a proof-of-concept implementation of restrict-and-relax search to successfully demonstrate its potential. To reach its full potential, more research is needed. There is clearly a need to investigate and explore alternative methods for producing the initial restricted integer program. The choice of the initial restricted integer program may be especially important for integer programs with few feasible solutions. Schemes that incorporate ideas from recent work on finding a small set of critical variables (a "backdoor" in the terminology of [12]) to be used first for branching, e.g., [9] and [6], may be quite powerful.

When encountering an infeasible LP, the generic implementation of restrict-and-relax search unfixes all previously fixed variables and resolves the LP to ascertain the status of the node and, if possible, to obtain dual information to guide the unfixing of variables. That is a pragmatic, but computationally expensive choice. When solving multicommodity fixed-charge network flow instances, we chose not to do this, but instead to use the dual information associated with LP solution at the parent node. There is a need to better understand the impact of these and other possible choices.

The generic implementation has a large number of parameters that control its behavior. We have observed that varying the parameter values can have significant impact on the performance on specific instances. The default values provide a compromise that seems to perform reasonably well across the instances in our test set. A better understanding of and better mechanisms for detecting when to restrict and when to relax during the search is essential.

An important component of a branch-and-bound algorithm is the node selection scheme. Node selection schemes, i.e., schemes that decide which of the active nodes to evaluate next, balance two goals: finding better solutions and proving that no better solutions exist. It is not clear that any of the node selection schemes currently employed by integer programming solvers is appropriate for restrict-and-relax search. The fact that it is possible to unfix variables at a node and thus locally enlarge the search space should be taken into account in node selection schemes.

To summarize, we have introduced a new branch-and-bound search paradigm for 0-1 mixed-integer programs that is designed to find high-quality solutions quickly, but is also capable of proving optimality or infeasibility. We have conducted computational experiments with a proof-of-concept implementation that indicates its potential, and have identified a number of research avenues that should make restrict-and-relax search even more powerful.

References

1. Berthold T (2012) RENS the optimal rounding. ZIP-Report 12–17, Konrad-Zuse-Zentrum fur Informationstechnik Berlin
2. Danna E, Rothberg E, Le Pape C (2005) Exploring relaxation induced neighborhoods to improve MIP solutions. Math Program 102:71–90
3. Dolan ED, More JJ (2002) Benchmarking optimization software with performance profiles. Math Program 91:201–213
4. Fischetti M, Glover F, Lodi A (2005) The feasibility pump. Math Program 104:91–104
5. Fischetti M, Lodi A (2003) Local branching. Math Program 98:23–47
6. Fischetti M, Monaci M (2011) Backdoor branching. In: Gunluk O, Woeginger GJ (eds) IPCO 2011. Springer, Berlin, pp 183–191
7. Hewitt M, Nemhauser GL, Savelsbergh MWP (2009) Combining exact and heuristic approaches for the capacitated fixed charge network flow problem. INFORMS J Comput 22:314–325
8. Hewitt M, Nemhauser GL, Savelsbergh MWP (2012) Branch-and-price guided search for integer programs with an application to the multicommodity fixed charge network flow problem. INFORMS J Comput. doi:10.1287/ijoc.1120.0503 (published online before print April 11, 2012)
9. Karzan F, Nemhauser GL, Savelsbergh MWP (2009) Information-based branching schemes for binary linear mixed integer problems. Math Program Comput 1:249–293
10. Koch T, Achterberg T, Andersen E, Bastert O, Berthold T, Bixby RE, Danna E, Gamrath G, Gleixner AM, Heinz S, Lodi A, Mittelmann H, Ralphs TK, Salvagnin D, Steffy DE, Wolter K (2011) MIPLIB 2010. Math Program Comput 3–2:103–163
11. Ralphs TK, Guzelsoy M (2005) The SYMPHONY callable library for mixed integer programming. In: Proceedings of the ninth INFORMS computing society conference, pp 61–76
12. Williams R, Gomes C, Selman B (2003) Backdoors to typical case complexity. In: Gottlob G, Walsh T (eds) IJCAI 2003: proceedings of the eighteenth international joint conference on artificial intelligence. Morgan Kaufmann, San Francisco, pp 1173–1178