DINS, a MIP Improvement Heuristic

Shubhashis Ghosh*

Department of Computing Science, University of Alberta, Canada
sghosh@ualberta.ca

Abstract. We introduce Distance Induced Neighbourhood Search (DINS), a MIP improvement heuristic that tries to find improved MIP feasible solutions from a given MIP feasible solution. DINS is based on a variation of local search that is embedded in an exact MIP solver, namely a branch-and-bound or a branch-and-cut MIP solver. The key idea is to use a distance metric between the linear programming relaxation optimal solution and the current MIP feasible solution to define search neighbourhoods at different nodes of the search tree generated by the exact solver. DINS considers each defined search neighbourhood as a new MIP problem and explores it by an exact MIP solver with a certain node limit. On a set of standard benchmark problems, DINS outperforms the MIP improvement heuristics Local Branching due to Fischetti and Lodi and Relaxation Induced Neighbourhood Search due to Danna, Rothberg, and Pape, as well as the generic commercial MIP solver Cplex.

1 Introduction

Mixed integer programs (MIPs) arise in many contexts; they are often intractable and NP-hard, even for feasibility [14]. Therefore, there is interest in designing effective heuristic methods for MIPs. Recently MIP heuristic development has specialized into finding better feasibility heuristic (that tries to find an initial MIP feasible solution), and improvement heuristic (that tries to find improved MIP feasible solutions from a given MIP feasible solution). In this paper, we present a new improvement heuristic.

Recent improvement heuristics such as Local Branching (LB), introduced by Fischetti et al. [9] and re-engineered by Danna et al. [5], and Relaxation Induced Neighbourhood Search (RINS), introduced by Danna et al. [5], work in tandem with a state-of-the-art exact solver such as Cplex MIP solver as follows. The exact solver generates a search tree using either branch-and-bound or branch-and-cut approach; the new heuristics periodically select nodes of the search tree at which to perform a localized search. Our heuristic also follows this approach. The heuristics differ primarily in the definition of the search neighbourhood; in LB the search neighbourhood is defined by restricting the number of 0-1 variables to switch their bounds from the known MIP feasible solution (referred as soft fixing), and in RINS it is defined by fixing some variables at their current values in the known MIP feasible solution (referred as hard fixing).

* The research support of NSERC is gratefully acknowledged.

Our search neighbourhood is defined in terms of a distance metric between a relaxation solution and the current MIP feasible solution, where the distance metric comes from the intuition that improved solutions are more likely to be close to the relaxation solution at the nodes of the search tree.

On a set of standard benchmark MIP instances, DINS outperforms Cplex, RINS, and LB with respect to the quality of solutions obtained within a time limit.

2 Related Previous Work

In order to show the strength of our heuristic, we compare it against Cplex, the exact solver in which it is embedded, and LB and RINS, the two recent improvement heuristics that are most similar in design.

Much research has been done in other kinds of MIP heuristics. There are several heuristics, introduced by Balas et al. [1], Faaland et al. [7], Hillier [12], and Ibaraki et al. [13], that incorporate some form of neighbourhood search, and most of them do so from the relaxation solution of MIP in order to find a MIP feasible solution.

There are also several pivot based heuristics, introduced by Balas et al. [2,3], Løkketangen et al. [15], Nediak et al. [18], and Løkketangen et al. [16], for MIP that try to obtain a MIP solution by performing pivots on the simplex tableau of the relaxation of MIP. Another heuristic introduced by Balas et al. [4], starting from the relaxation solution of MIP, tries to find a MIP solution by first using some pivoting on the simplex tableau and then doing some form of neighbourhood search. Recently Fischetti et al. [8] introduce another heuristic to find a MIP solution from the relaxation solution of MIP, where they solve a sequence of linear programs in the process of finding a MIP feasible solution.

3 Methods

We assume that the input program P is a generic MIP of the form shown below, where c, x, b, A have dimensions $n, n, m, m \times n$ respectively, $N = \{1, \ldots, n\}$ is the set of variable indices of P which is partitioned into $(\mathcal{B}, \mathcal{G}, \mathcal{C})$ with $\mathcal{B}$, $\mathcal{G}$, and $\mathcal{C}$ denoting the indices of 0-1, general integer, and continuous variables respectively. An integer variable is any variable in $\mathcal{B} \cup \mathcal{G}$.

$$\begin{aligned} P : min\ \{\ & c^T x \mid Ax \geq b,\ x_i \in \{0,1\}\ \forall i \in \mathcal{B}, \\ & x_j \geq 0 \text{ and integer } \forall j \in \mathcal{G},\ x_j \geq 0\ \forall j \in \mathcal{C}\} \end{aligned}$$

Since we compare DINS with LB and RINS, we describe LB and RINS in some details.

3.1 Local Branching

LB defines the neighbourhood of a feasible solution x^* by limiting at some integer p the number of 0-1 variables currently at 0 or 1 that can switch their bounds. This is achieved by adding to the instance the *LB inequality* $D(x, x^*) \leq p$, where

$$D(x, x^*) := \sum_{j \in V_0} x_j + \sum_{j \in V_1} (1 - x_j),$$

and where V_0 and V_1 are the index sets of the 0-1 variables that are at 0 and 1 respectively in x^*.

LB has been implemented in two different ways. Originally, Fischetti and Lodi [9] treated it as an external branching framework (i.e., creates branches in the search tree by $D(x, x^*) \leq p$ and $D(x, x^*) \geq p + 1$ as opposed to the standard branching which are done on the variables in the branch-and-bound framework) in addition to an heuristic and obtained the diversification (i.e., switching the search in a different region of the MIP feasible space) by defining the neighbourhoods with a change in the value of the parameter p. Later, Danna et al. [5] implemented LB solely as a heuristic and obtained the diversification by defining the neighbourhoods on the new solutions found during the MIP search tree exploration. Danna et al. showed that their implementation of LB outperformed the original. For this reason, we choose the Danna et al. version of LB to compare against our DINS.

3.2 Relaxation Induced Neighbourhood Search

During the exploration of the MIP search tree, the relaxation solution at successive nodes (that are not pruned by infeasibility or bound) provides a better objective value than the objective value of the current MIP solution. Using this, Danna et al. introduce RINS making the intuition that, in improved MIP solutions, it is more likely for the variables to stay at the same values those agree in the current MIP solution and current node relaxation solution. Thus RINS defines the promising neighbourhood fixing all variables whose values at the current MIP solution are equal to their respective values at the current node relaxation solution.

In the implementation of RINS[1], the procedure for exploring the RINS defined neighbourhood is invoked at a particular node of the MIP search tree. At the termination of the procedure, the MIP search tree is resumed, and if the procedure finds a new MIP solution, the MIP solution at the MIP search tree is updated.

As noted by Danna et al. in [5], consecutive nodes of the MIP search tree provide almost identical relaxation solution. Therefore, the RINS procedure is called only every f nodes for some reasonably large f.

3.3 Distance Induced Neighbourhood Search

In contrast to RINS, which performs only hard fixing of variables, and LB, which performs only soft fixing of variables, our DINS incorporates some hard fixing, some soft fixing, and some rebounding (changing lower and upper bounds

[1] ILOG Cplex 9.13 comes with an implementation of RINS and can be invoked by setting the Cplex parameter *IloCplex::MIPEmphasis* to 4 [5].

of the variables), all based on a distance metric. In the next sections we show that DINS outperforms both RINS and LB[2] on an instance test bed that includes all the instances studied in [5,9] as well as some other hard instances from other sources.

Like RINS, DINS also rely on the fact that, during exploring the MIP search tree, the relaxation solution at successive nodes (those are not pruned by infeasibility or bound) provides a better objective value compared to the objective value provided by the current MIP solution.

But unlike RINS, the intuition in DINS is that the improved MIP solutions are more likely to be the close ones to the current relaxation solution. An exact modeling of this intuition would require inclusion of the following quadratic inequality which unfortunately cannot be expressed as a linear constraint.

$$\sum_{j \in N} (x_j - x_{j(node)})^2 \leq \sum_{j \in N} (x_{j(mip)} - x_{j(node)})^2,$$

where x_{mip} and x_{node} denote the current MIP solution and the current relaxation solution, and for a variable x_j, $x_{j(mip)}$ and $x_{j(node)}$ denote the values of x_j in x_{mip} and x_{node} respectively.

DINS relaxes the intuition by considering that the improved MIP solutions are close to x_{node} only with respect to the integer variables and choosing the following inequality based on absolute differences as the measure of close ones.

$$\sum_{j \in \mathcal{B} \cup \mathcal{G}} |x_j - x_{j(node)}| \leq \sum_{j \in \mathcal{B} \cup \mathcal{G}} |x_{j(mip)} - x_{j(node)}|.$$

DINS then partially captures this inequality (the chosen distance metric) by defining a neighbourhood with some rebounding, some hard fixing, and some soft fixing of the integer variables.

We notice that if an integer variable x_j, for which the absolute difference, $|x_{j(mip)} - x_{j(node)}|$, is less than 0.5, takes a different value than $x_{j(mip)}$ in an improved solution, the absolute difference increases. On the contrary, if an integer variable, for which the absolute difference is greater or equal to 0.5, takes a different value than $x_{j(mip)}$ in an improved solution, the absolute difference may not increase.

DINS computes new lower and upper bounds of an integer variable x_j, for which the absolute difference is greater or equal to 0.5, so that at an improved solution the absolute difference does not increase. Considering l_j^{old} and u_j^{old} as the existing lower and upper bounds of x_j, DINS computes the new lower and upper bound l_j^{new} and u_j^{new} respectively as follows:

```
if $(x_{j(mip)} \geq x_{j(node)})$ then
    $l_j^{new} \leftarrow \max(l_j^{old}, \lceil x_{j(node)} - (x_{j(mip)} - x_{j(node)}) \rceil)$,   $u_j^{new} \leftarrow x_{j(mip)}$
elsif $(x_{j(mip)} < x_{j(node)})$ then
    $l_j^{new} \leftarrow x_{j(mip)}$,   $u_j^{new} \leftarrow \min(u_j^{old}, \lfloor x_{j(node)} + (x_{j(node)} - x_{j(mip)}) \rfloor)$.
```

[2] In [5], Danna et al. have tried two hybrid strategies of RINS and LB and concluded that their performance were not better than RINS alone.

We refer it as rebounding; the rebounding does not change existing bounds for all the variables that fall in this category (for example, no 0-1 variable in this category change its bounds). If all the integer variables, for which $|x_{j(mip)} - x_{j(node)}| < 0.5$, are fixed to their respective current values, then any solution found from this neighbourhood exploration will obviously be a closer one to x_{node} in terms of the chosen distance metric. But the sum of absolute differences can also decrease if the total decrease d in the sum of absolute differences caused by the integer variables for which $|x_{j(mip)} - x_{j(node)}| \geq 0.5$ is greater than the total increase d' in the sum of absolute differences caused by the integer variables for which $|x_{j(mip)} - x_{j(node)}| < 0.5$.

DINS partially captures this observation by allowing the integer variables x_j, for which $|x_{j(mip)} - x_{j(node)}| < 0.5$, to change their values in x_{mip} so that d' is not larger than a chosen small number p. It does this by performing some soft fixing and some hard fixing of these variables. DINS performs soft fixing through the LB inequality which requires introduction of new variables when general integer variables are considered. As in [9] and [5], DINS constructs LB inequality using only 0-1 variables. Therefore, all the general integer variables x_j with $|x_{j(mip)} - x_{j(node)}| < 0.5$ are fixed (hard fixing) at $x_{j(mip)}$.

Among the 0-1 variables with $|x_{j(mip)} - x_{j(node)}| < 0.5$, DINS performs some hard fixing like RINS, but incorporates some more intuition in this process. Like RINS, DINS chooses the same set of variables, that agree in both the current MIP solution and the current node relaxation solution, as the primary candidates for hard fixing. Then it applies a filtering step to this primary candidate set using two information. First information comes from the intuition that if an integer variable, in the primary candidate set, takes the same value in the root relaxation solution of MIP search tree and current node relaxation solution, is more likely to take the same value in improved MIP feasible solutions. The second information comes from the intuition that if an integer variable, in the primary candidate set, takes the same value in the previously encountered MIP solutions, is more likely to take the same value in improved MIP feasible solutions. This two information actually gather knowledge from both the relaxation solutions and previously encountered MIP solutions. DINS uses an array of flag for the integer variables to keep track which variables have taken different values in the previously encountered MIP solutions. Thus the hard fixing in DINS can be stated more explicitly in the following way: let x_{mip}, x_{node}, and x_{root} denote the current MIP solution, the current node relaxation solution, and the root relaxation solution respectively. Also let Δ is an array where $\Delta[j]$ is set if x_j has taken different values in previously encountered MIP solutions. Therefore, a variable x_j is fixed (hard fixing) at value $x_{j(mip)}$ if $x_{j(mip)} = x_{j(node)} = x_{j(root)}$ and $\Delta[j]$ is clear.

Consider $\mathcal{F}$ and $\mathcal{H}$ denote the set of variables for which rebounding and hard fixing has been performed respectively. Now assume $\mathcal{R}$ be the set of variables where $\mathcal{R} = (\mathcal{B} \cup \mathcal{G}) - \mathcal{F} - \mathcal{H}$. According to our construction $\mathcal{R}$ contains only 0-1 variables.

DINS now performs soft fixing on the variables in $\mathcal{R}$, when $|\mathcal{R}| \neq \phi$, by adding the following LB inequality:

$$\sum_{j \in \mathcal{R} \,\wedge\, x_{j(mip)}=0} x_j + \sum_{j \in \mathcal{R} \,\wedge\, x_{j(mip)}=1} (1 - x_j) \;\leq\; p$$

As noted earlier, our intuition is that improved feasible solutions are more likely to be obtained by getting close to the current relaxation solution from the current MIP solution. Therefore, DINS generates the promising neighbourhood taking small value for p which means that a solution, in this defined neighbourhood, can have a sum of absolute differences increased by at most p.

Whenever DINS procedure is invoked at a particular node of MIP search tree, it creates the described neighborhood with the initial chosen value of p and explores it using a branch-and-bound or a branch-and-cut solver with a specified node limit nl. If the exploration reaches the node limit without finding a new solution, DINS reduces p by 5 and explores a new neighbourhood. This continues until $p < 0$, or the neighbourhood exploration finds a new solution or the neighbourhood is explored completely without finding a new solution. Whenever the neighbourhood exploration finds a new solution, p is reset to its initial chosen value and continues in the same fashion. The procedure in Figure 1 describes the operation sequence of DINS at a particular node of the MIP search tree. At the termination of the procedure, the MIP search tree is resumed and, if the procedure finds a new MIP solution, the MIP solution at the MIP search tree is updated.

Like RINS, the DINS procedure is called first when the MIP search tree finds its first MIP solution and, thereafter, at every f nodes of the MIP search tree.

4 Computational Results

4.1 Experimental Setup and Instance Test Bed

We implement LB, RINS, and DINS in the C programming language with the MIP search tree generated by Cplex 9.13 MIP solver. All experiments are run on an 2403 MHz AMD Athlon processor with 128 MByte of memory under Redhat Linux 9.0. An implementation of DINS is available at [11].

We compose a benchmark test bed of MIP instances with the property that the test bed excludes the instances which default Cplex either solves to optimality or fails to find a MIP solution in one CPU-hour. With this criteria we have 64 MIP instances (all have some 0-1 variables), described in [10], from the following sources commonly used as benchmark instances for MIP solvers.

– Twenty six instances used in the local branching paper [9]. These instances have been collected from the instance set maintained by DEIS operations research group [6].
– Twelve more instances from the instance set maintained by DEIS operations research group [6].

- Eleven instances from MIPLIB 2003 [17].
- Five job-shop scheduling instances with earliness and tardiness costs used in [8].
- Eleven network design and multi-commodity routing instances used in [5].

```
Procedure DINS_at_tree_node
INPUT: a 0-1 mixed integer problem $P$, the current MIP solution $x_{mip}$,
    the current node relaxation solution $x_{node}$, the root relaxation solution $x_{root}$,
    parameter $p$, node limit $nl$, and the flag array $\Delta$.
OUTPUT: A new MIP solution $x^*$ ($x_{mip}$ in case of failure in finding a new solution).

1.  if ($x_{mip}$ is a new MIP solution compared to the MIP solution
        at the termination of last call of this procedure)
            update the array $\Delta$ accordingly
2.  $x^* \leftarrow x_{mip}$, $p_{current} \leftarrow p$, exploreAndNoSolution $\leftarrow$false
3.  repeat
4.      construct $P+$ from $P$ as follows:
            (i) perform rebounding on the variables $x_j$ for which $|x_j^* - x_{j(node)}| \geq 0.5$,
            (ii) perform hard fixing of the general integer variables $x_j$ for which
                $|x_j^* - x_{j(node)}| < 0.5$,
            (iii) perform hard fixing of the 0-1 integer variables $x_j$ for which
                $|x_j^* - x_{j(node)}| < 0.5$ and $x_j^* = x_{j(node)} = x_{j(root)}$ and $\Delta[j]$ is clear,
            (iv) let $\mathcal{R}$ be the set of remaining 0-1 integer variables.
                if ($\mathcal{R} \neq \phi$) perform soft fixing by adding the inequality
                $\sum_{j \in \mathcal{R} \wedge x_j^*=0} x_j + \sum_{j \in \mathcal{R} \wedge x_j^*=1} (1 - x_j) \leq p$
5.      Apply black-box MIP solver to $P+$ with node limit $nl$ and
            an objective cutoff equal to the objective value provided by $x^*$
6.      if (a new solution $x_{new}$ is obtained) then
7.          $x^* \leftarrow x_{new}$, $p_{current} \leftarrow p$, update the array $\Delta$
8.      elsif (node limit reached without having a new solution) then
9.          if($|\mathcal{R}| = \phi$) $p_{current} = -1$
10          else $p_{current} \leftarrow p_{current} - 5$
11.     else exploreAndNoSolution $\leftarrow$true
12. until ($p_{current} < 0$ or exploreAndNoSolution)
13. return $x^*$
```

Fig. 1. Procedure DINS_at_tree_node

4.2 Comparison Among Methods

We compare DINS against RINS, LB, and Cplex in its default setup (default Cplex). One CPU-hour is set to be the execution time for each method and it seems to be sufficient to distinguish the effectiveness of all the methods.

Default Cplex is used for exploring the neighbourhoods generated in LB, RINS, and DINS. The three methods namely LB, RINS, and DINS have a set of parameters which need to be set. As used in [5], for LB, we set $p = 10$ and $nl = 1000$, and for RINS, we use Cplex 9.13 with the parameter *IloCplex::MIPEmphasis* set to 4 where, according to [5], $f = 100$ and $nl = 1000$. For

DINS, we set $p = 5$ (different from LB to relax our intuition a little as well as to make the neighbourhood small), $f = 100$ and $nl = 1000$.

Following Danna et al. [5], we carry out two set of experiments; in one set of experiments we invoke all four methods with a presumably poor solution at the root node of the MIP search tree, and in the other we invoke all four methods with a presumably good solution at the root node of the MIP search tree. Although there is no exact way to distinguish a good and a bad MIP solution, following Danna et al. [5], we presume that the first MIP solution found by the default Cplex MIP solver represents a poor solution, and the solution obtained by default Cplex in one CPU-hour represents a good solution.

In order to capture the quality of obtained solution by each method, we use the measure *percentage of gap* defined by 100*|(obj. value of obtained solution - obj. value of the best known solution) /obj. value of the best known solution|. Table 1 and Table 2 show the percentage of gap obtained at the end of one CPU-hour by all the four methods considered in this paper, where the bold face identifies the best method for the corresponding instance (multiple bold faces appear if there are multiple methods obtaining the same solution).

Following Danna et al. [5], we group the instances into three different sets so that the effectiveness of different methods in different groups becomes visible. According to [5], the groups are defined as 'small spread', 'medium spread', and 'large spread' instances where the gap between the worst solution found by any of the four methods considered in this paper and the best known solution is less than 10%, between 10% and 100%, and larger than 100% respectively. The percentage of gap shown in Table 1 and Table 2 are used to group the instances.

We use three measures to evaluate the performance of different methods.

Our first measure is *best in number of instances*, which represents the number of instances at which a method finds the best solution among the solutions obtained by all the four methods. If multiple methods find the same best solution for an instance, then the instance contributes one in the measures for all the corresponding methods.

Our second measure is the *average percentage of gap*, which represents the arithmetic mean of the percentage of gaps obtained by a method on a group of instances at a certain point of execution.

Our third measure is the *average percentage of improvement*, which represents the arithmetic mean of percentage of improvements obtained by a method on a group of instances at a certain point of execution. In order to visualize how much improvement has been obtained by different methods starting from a presumably poor and good solution, we define the *percentage of improvement* for an instance as 100*|(obj. value of the initial solution - obj. value of the obtained solution) /obj. value of the initial solution|.

Table 3 represents the comparative results of four different methods for both set of experiments.

As expected, DINS, comparing against all other three methods in both set of experiments, has higher percentage of improvement and lower percentage of gap for each of the categorized group of instances, and obtains best solution in

Table 1. Percentage of Gap = 100 $*$ |(obj. value of obtained solution - obj. value of the best known solution)/obj. value of the best known solution| in one CPU-hour

problem	Percentage of Gap			
	Default Cplex	LB	RINS	DINS
Small spread instances				
a1c1s1	2.347	0.250	**0.000**	0.079
a2c1s1	2.978	1.889	**0.000**	0.024
b1c1s1	5.977	1.786	**0.933**	4.444
b2c1s1	4.240	2.701	**0.559**	1.010
biella1	**0.309**	0.806	0.426	0.739
danoint	**0.000**	**0.000**	**0.000**	**0.000**
mkc	0.180	0.049	0.043	**0.021**
net12	**0.000**	**0.000**	**0.000**	**0.000**
nsrand-ipx	0.625	0.625	0.313	**0.000**
rail507	**0.000**	**0.000**	**0.000**	**0.000**
rail2586c	2.518	2.204	1.994	**1.574**
rail4284c	1.774	1.867	**1.027**	**1.027**
rail4872c	1.742	1.290	1.097	**1.032**
seymour	0.473	0.473	**0.000**	0.236
sp97ar	0.428	0.513	0.335	**0.000**
sp97ic	0.793	0.642	0.551	**0.000**
sp98ar	0.184	**0.106**	0.177	0.228
sp98ic	0.270	0.146	0.204	**0.072**
tr12-30	**0.000**	0.024	**0.000**	**0.000**
arki001	0.003	0.003	0.004	**0.002**
roll3000	0.543	0.303	**0.070**	**0.070**
umts	0.013	0.049	0.022	**0.002**
berlin-5-8-0	**0.000**	**0.000**	**0.000**	**0.000**
bg512142	7.257	5.192	0.161	**0.000**
blp-ic97	0.779	0.653	0.358	**0.000**
blp-ic98	0.961	1.056	0.746	**0.515**
blp-ar98	0.655	0.060	0.461	**0.000**
cms750-4	2.372	**0.791**	1.186	**0.791**
dc1l	2.018	8.166	6.994	**1.572**
railway-8-1-0	0.250	**0.000**	0.250	0.250
usabbrv-8-25-70	3.306	2.479	**0.000**	1.653
aflow40b	0.257	1.455	**0.000**	**0.000**
dano3mip	2.602	3.595	4.724	**2.230**
fast0507	**0.000**	0.575	0.575	**0.000**
harp2	0.001	0.001	0.023	**0.000**
t1717	7.948	**1.939**	5.979	7.948
noswot	**0.000**	**0.000**	**0.000**	**0.000**
timtab1	7.469	7.779	**0.000**	**0.000**
ljb2	**0.256**	3.329	1.576	3.329
rococoB10-011000	0.802	2.848	**0.437**	**0.437**
rococoB11-010000	5.039	5.839	**1.768**	2.196
rococoB12-111111	5.204	4.489	3.738	**2.541**

Table 2. Percentage of Gap = 100 ∗ |(obj. value of obtained solution - obj. value of the best known solution)/obj. value of the best known solution| in one CPU-hour

Continued from Table 1				
problem	Percentage of Gap			
	Default Cplex	LB	RINS	DINS
Small spread instances				
rococoC10-001000	0.044	0.113	0.044	**0.000**
rococoC11-011100	6.018	9.991	9.244	**5.879**
rococoC12-111100	5.188	5.188	**1.298**	4.016
Medium spread instances				
glass4	13.014	7.534	**2.740**	4.794
swath	18.067	5.679	8.089	**4.622**
dg012142	17.457	25.984	4.963	**3.943**
liu	**2.475**	10.066	3.465	5.281
timtab2	16.373	18.484	3.188	**0.912**
ljb7	7.424	21.834	**4.367**	8.908
ljb9	50.717	70.866	55.074	**50.690**
ljb10	**0.807**	13.929	13.693	8.578
rococoB10-011001	7.660	5.309	**5.220**	10.082
rococoB11-110001	9.994	19.558	**4.267**	6.894
rococoC10-100001	16.041	**7.387**	13.316	10.070
rococoC11-010100	27.431	13.615	10.546	**9.029**
rococoC12-100000	12.928	10.090	5.623	**2.799**
Large spread instances				
markshare1	500.000	**400.00**	**400.00**	500.00
markshare2	1300.000	**1100.000**	2000.000	1800.000
dc1c	695.213	2.353	**0.296**	0.773
trento1	**0.000**	193.118	1.912	0.402
ds	11.226	945.745	11.226	**6.119**
ljb12	**39.273**	323.183	49.599	64.987

higher number of instances. It is to be noted that, starting from a presumably good solution, DINS has become best in more number of instances than the number of instances in which it has been best in the experimentation with bad solution.

Furthermore, for different group of instances in Figure 2– 4, we sketch how different methods improve the solution quality (average percentage of gap) over time starting from presumably poor solutions. We can draw some basic conclusions analyzing these figures. For all three group of instances, DINS performance is worse comparing to that of RINS at the initial level of computation, but DINS performance becomes better as the computation progresses and once it becomes better, it maintains its lead over RINS for the remaining part of the computation. For small and large spread instances, DINS obtains the lead over RINS earlier than in medium spread instances. Similarly in medium and large spread instances, DINS performance is worse comparing to that of default Cplex at the initial level of computation, but DINS outperforms default Cplex as the

Table 3. A comparative performance summary for different methods

Average % of improvement				
Group of Instances (# of instances)	Default Cplex	LB	RINS	DINS
experiments from the presumably poor solutions				
all instances (64)	36.19	35.49	38.01	38.05
small spread (45)	23.41	23.61	23.90	23.92
medium spread (13)	60.43	60.25	62.05	62.29
large spread (6)	80.78	70.90	91.64	91.66
experiments from the presumably good solutions				
all instances (64)	2.35	3.04	3.45	3.96
small spread (45)	0.45	0.78	1.26	1.29
medium spread (13)	2.50	4.91	5.10	6.57
large spread (6)	16.31	15.96	16.27	18.47

Average % of gap				
Group of Instances (# of instances)	Default Cplex	LB	RINS	DINS
experiments from the presumably poor solutions				
all instances (64)	44.22	51.19	41.33	39.73
small spread (45)	1.86	1.81	1.05	0.97
medium spread (13)	15.41	17.72	10.35	9.74
large spread (6)	424.28	494.07	410.51	395.38
experiments from the presumably good solutions				
all instances (64)	32.43	31.67	31.21	29.14
small spread (45)	1.41	1.07	0.56	0.54
medium spread (13)	13.57	10.63	10.46	8.59
large spread (6)	305.92	306.77	306.06	288.17

Best in # of instances				
Group of Instances (# of instances)	Default Cplex	LB	RINS	DINS
experiments from the presumably poor solutions				
all instances (64)	13	12	25	39
small spread (45)	9	9	19	32
medium spread (13)	2	1	4	6
large spread (6)	2	2	2	1
experiments from the presumably good solutions				
all instances (64)	16	23	29	48
small spread (45)	13	17	26	35
medium spread (13)	1	5	2	8
large spread (6)	2	1	1	5

computation progresses. LB is always worse than RINS and DINS where, at the end of time limit, LB has an edge over default Cplex only in small spread instances.

In an attempt to see how good intuition DINS has made, we provide some statistical measures from our experimental results. It has been seen that, the

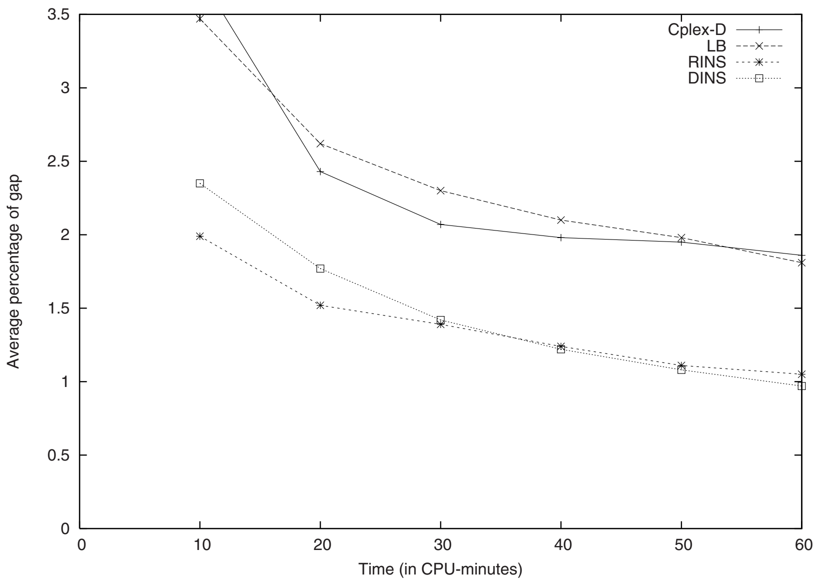

Fig. 2. progress of different methods in reducing percentage of gap on the 45 small spread instances

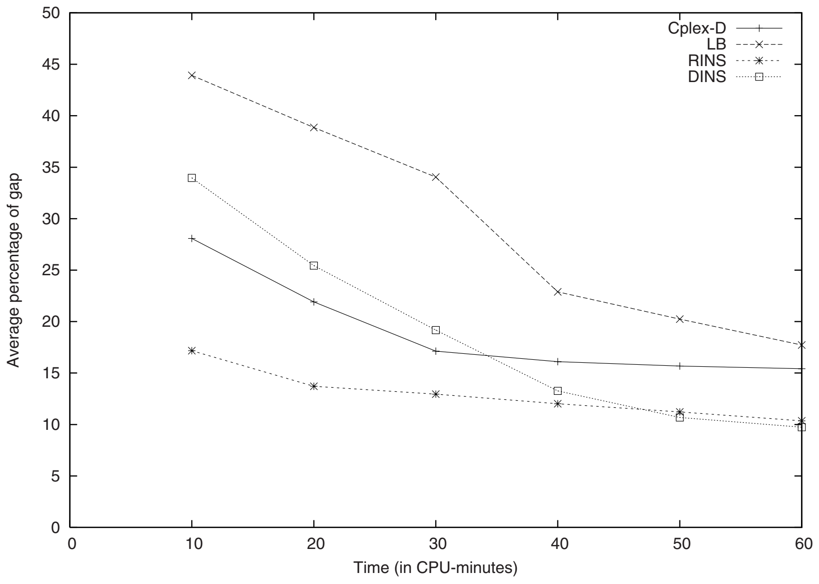

Fig. 3. progress of different methods in reducing percentage of gap on the 13 medium spread instances

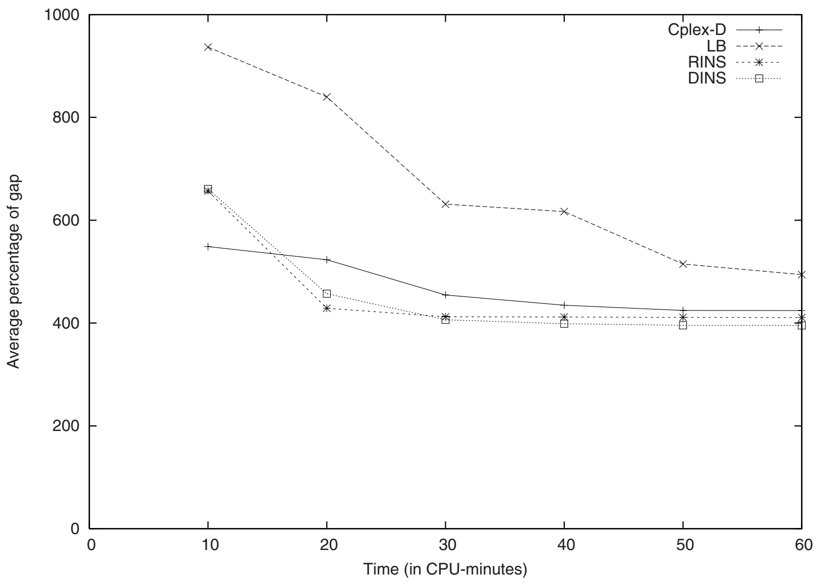

Fig. 4. progress of different methods in reducing percentage of gap on the 6 large spread instances

number of times neighbourhood exploration finds a new solution in all the instances, the chosen distance metric was satisfied in 80.89% occurrences, and the quadratic distance metric was satisfied in 80.5% occurrences. These experimental results support our intuition that improved solutions are more likely to be close to the node relaxation solutions, and also support our choice of distance metric. Moreover, relaxing the chosen distance metric a little bit gives DINS the extra power of finding those improved solutions that do not satisfy the chosen distance metric at the node at which the solution has been obtained, but probably would satisfy the chosen distance metric at some deeper nodes of the MIP search tree.

5 Conclusions

We have introduced DINS, a heuristic to find improved MIP feasible solutions from a known MIP feasible solution, based on a distance metric between the current MIP solution and the current node relaxation solution.

A comparison of DINS against existing neighbourhood search based heuristics shows that it outperforms both RINS and LB in obtaining good MIP solutions within a certain time limit and in the power of improving both poor and good MIP solutions.

Unlike RINS, DINS uses the change of relaxation solution between the root and the node and the change in the encountered MIP solutions in guiding the hard fixing of 0-1 variables; this has an effect in finding the good MIP solutions as the computation progresses. This has been experimentally visualized by having a comparatively worse performance on the benchmark instances by running a modified DINS where the hard fixing of 0-1 variables are carried out according to the hard fixing of RINS.

Acknowledgements. We thank Emilie Danna for the useful email discussions during the implementation and analysis of the methods.

References

1. E. Balas, S. Ceria, M. Dawande, F. Margot, and G. Pataki. Octane: a new heuristic for pure 0-1 programs. *Operations Research*, 49(2):207–225, 2001.
2. E. Balas and C.H. Martin. Pivot and complement – a heuristic for 0-1 programming. *Management Science*, 26(1):86–96, 1980.
3. E. Balas and C.H. Martin. Pivot and shift – a heuristic for mixed integer programming. Technical report, GSIA, Carnegie Mellon University, 1986.
4. E. Balas, S. Schmieta, and C. Wallace. Pivot and shift – a mixed integer programming heuristic. *Discrete Optimization*, 1:3–12, 2004.
5. E. Danna, E. Rothberg, and C.L. Pape. Exploring relaxation induced neighborhhods to improve mip solutions. *Mathematical Programming*, 102:71–90, 2005.
6. DEIS. Library of instances. www.or.deis.unibo.it/research_pages/ORinstances/.
7. B.H. Faaland and F.S. Hillier. Interior path methods for heuristic integer programming procedures. *Operations Research*, 27(6):1069–1087, 1979.
8. M. Fischetti, F. Glover, and A. Lodi. The feasibilty pump. to be appeared on Mathematical Programming.
9. M. Fischetti and A. Lodi. Local branching. *Mathematical Programming B*, 98: 23–49, 2003.
10. S. Ghosh. Description of all the used benchmark instances in this paper. www.cs.ualberta.ca/∼shubhash/dins/benchmarks.ps.
11. S. Ghosh. Implementation of DINS. www.cs.ualberta.ca/∼shubhash/codes.html.
12. F.S. Hillier. Efficient heuristic procedures for integer linear programming with an interior. *Operations Research*, 17(4):600–637, 1969.
13. T. Ibaraki, T. Ohashi, and H. Mine. A heuristic algorithm for mixed-integer programming problems. *Math. Program. Study.*, 2:115–136, 1974.
14. R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.
15. A. Løkketangen, K. Jörnsten, and S. Storøy. Tabu search within a pivot and complement framework. *International Transactions in Operational Research*, 1(3): 305–317, 1994.
16. A. Løkketangen and D.L. Woodruff. Integrating pivot based search with branch and bound for binary mip's. *Control and Cybernetics, Special issue on Tabu Search*, 29(3):741–760, 2001.
17. A. Martin, T. Achterberg, and T. Koch. Miplib 2003. http://miplib.zib.de.
18. M. Nediak and J. Eckstein. Pivot, cut, and dive: A heuristic for mixed 0-1 integer programming. *RUTCOR Research Report*, RRR 53-2001, 2001.