J Heuristics (2014) 20:589–615
DOI 10.1007/s10732-014-9255-0

Boosting local search with Lagrangian relaxation

Zhilei Ren · He Jiang · Shuwei Zhang · Jingxuan Zhang · Zhongxuan Luo

Received: 2 November 2013 / Revised: 18 May 2014 / Accepted: 1 July 2014 /
Published online: 17 July 2014

Abstract Local search algorithms play an essential role in solving large-scale combinatorial optimization problems. Traditionally, the local search procedure is guided mainly by the objective function of the problem. Hence, the greedy improvement paradigm poses the potential threat of prematurely getting trapped in low quality attraction basins. In this study, we intend to utilize the information extracted from the relaxed problem, to enhance the performance of the local search process. Considering the Lin-Kernighan-based local search (LK-search) for the p-median problem as a case study, we propose the Lagrangian relaxation Assisted Neighborhood Search (LANS). In the proposed algorithm, two new mechanisms, namely the neighborhood reduction and the redundancy detection, are developed. The two mechanisms exploit the information gathered from the relaxed problem, to avoid the search from prematurely targeting low quality directions, and to cut off the non-promising searching procedure, respectively. Extensive numerical results over the benchmark instances demonstrate that LANS performs favorably to LK-search, which is among the state-of-the-art local search algorithms for the p-median problem. Furthermore, by embedding LANS into other heuristics, the best known upper bounds over several benchmark instances could be updated. Besides, run-time distribution analysis is also employed to investigate the reason why LANS works. The findings of this study confirm that the idea of improving local search by leveraging the information induced from relaxed problem is feasible and practical, and might be generalized to a broad class of combinatorial optimization problems.

Electronic supplementary material The online version of this article (doi:10.1007/s10732-014-9255-0) contains supplementary material, which is available to authorized users.

Z. Ren · H. Jiang (✉) · S. Zhang · J. Zhang · Z. Luo
School of Software, Dalian University of Technology, Dalian, China
e-mail: jianghe@dlut.edu.cn

Keywords Local search · p-median · Lagrangian relaxation · Lin-Kernighan neighborhood

1 Introduction

Local search is among the most prominent and widely applied techniques for solving large-scale combinatorial optimization problems, as either a stand-alone algorithm, or an embedded subroutine within metaheuristics (Resende and Werneck 2004; Vela et al. 2010; Brimberg et al. 2008; Ceschia and Schaerf 2013; Riise and Burke 2011; Gutin and Karapetyan 2009; Brueggemann and Hurink 2011). For example, the LKH heuristic (Helsgaun 2009) represents the state-of-the-art local search algorithm to solve the Traveling Salesman Problem (TSP), which is able to achieve optimal solutions over large-scale instances with up to 85,900 variables. In the domain of Satisfiability (SAT) problem, heuristics such as CRSat (Belov et al. 2011) and its variants are able to achieve highly effective performances over very large-scale instances. These algorithms are usually classified as stochastic local search (Hoos 2005), in which local search constitutes the core component.

Starting from an initial solution, local search iteratively traverses the neighborhood of the incumbent solution, in search of solutions with better quality.[1] Traditionally, local search is mainly guided by the objective evaluation. Hence, there is usually no sufficient information to measure how much a local optimum deviates from the global optimum during the search. Thus, it would be ideal if information indicating the relative quality of the current position could be incorporated in local search. With the guidance of the introduced information, we may be able to avoid the situation in which local search gets stuck in poor local optimum trap, and might be able to detect the non-promising search regions of the solution space, and terminate the redundant search operations.

Meanwhile, recent years have witnessed the success of emerging trends and new approaches that leverage the information induced from relaxed problem to guide the heuristic search procedure. These approaches have been demonstrated to be very effective. However, the information exploitation is mostly realized at the global level, such as sorting the neighborhood order (Puchinger and Raidl 2008) and search space reduction (Ren et al. 2012a). To the best of our knowledge, the issue of guiding the search at the local level with the relaxation information has not yet been systematically investigated.

Considering the Lin-Kernighan-based local search (LK-search) for the p-median problem as a case study, we intend to employ the Lagrangian relaxation to guide the local search algorithm. To achieve this, we propose the Lagrangian relaxation Assisted Neighborhood Search (LANS). LANS enhances the performance of LK-search from two aspects. On the one hand, to avoid the search from prematurely targeting low quality directions, we develop a neighborhood reduction mechanism, which works by

[1] Sometimes the algorithms that combine local search with stochastic perturbations are also loosely denoted as local search. However, in this study, we only refer to local search as those algorithms that traverse neighborhoods deterministically.

eliminating those variables that may lead to increased lower bounds, from the candidate sets. This mechanism is inspired by the branching rules that are widely adopted in tree search such as beam search (Croce et al. 2004; Bennell and Song 2010) and local branching (Fischetti and Lodi 2003). On the other hand, we propose a redundancy detection mechanism, to detect and cut off the non-promising searching procedure, which is motivated by the bounding operation of branch-and-bound (Järvinen et al. 1972; Christofides and Beasley 1982; Li and Quan 2010).

In order to evaluate the performance of LANS, extensive experimental study is conducted. Numerical results over a series of benchmark instances demonstrate that LANS performs favorably to LK-search, which is among the state-of-the-art local search algorithms for the p-median problem. Then, by embedding LANS as a subroutine, we examine the behavior of LANS in the context of metaheuristics. Finally, the run-time distribution analysis is carried out, over LANS, LK-search, and two other variant versions of LK-search, to investigate why LANS works. Using the run-time distribution analysis, we observe that both the neighborhood reduction and the redundancy detection mechanisms achieve the expected objectives, i.e., the neighborhood reduction mechanism is able to prevent the search from prematurely getting trapped in low quality attraction basin, and the redundancy detection mechanism monitors the estimated quality of the search, and could terminate the non-promising search process. Furthermore, by integrating the two mechanisms with LK-search, the proposed LANS shows the dominance in the analysis, in terms of both the effectiveness and the efficiency.

The contributions of this paper could be summarized as follows. (1) We propose a neighborhood reduction mechanism, to avoid the local search from being prematurely trapped in low quality attraction basins. (2) We develop a redundancy detection method, to detect and terminate the neighborhood traversal that could not find promising solutions. (3) Extensive experiments demonstrate both the effectiveness and the efficiency of the proposed local search. In particular, by embedding LANS into other heuristic frameworks, several best known results over large-scale Euclidean benchmark instances could be updated.

The rest of the paper is organized as follows. In Sect. 2, we briefly review the related work in the literature. In Sect. 3, we present the Lagrangian relaxation Assisted Neighborhood Search. Then, extensive experimental results are reported and discussed in Sect. 4. Finally, in Sect. 5, we conclude the paper, and point out several potential directions for the future work.

2 Background

2.1 Local search algorithms for the p-median problem

In this subsection, we introduce the basic information of the p-median problem, as well as the local search algorithms for solving the problem. Given a set F of m facilities, a set U of n users, a distance matrix D (d_{ij} indicates the distance between user i and facility j), and a constant $p \leq m$, the p-median problem is to select a median set $J \subseteq F$, $|J| = p$, to minimize the sum of the distances from each user to his/her

nearest median. It is straightforward that, solutions could be encoded as fixed length subsets of the facility set.

As one of the basic problems in the field of location science, the p-median problem has attracted great research attention since it was first issued in the 1960s (Hakimi 1964), due to its wide applications to various domains (Reese 2006; Mladenovic et al. 2007). Various algorithms have been adopted to solve the problem (Hansen and Mladenovic 1997; Resende and Werneck 2004; Avella et al. 2007; García et al. 2011; Ren et al. 2012a). However, because of the intrinsic difficulty of the p-median problem, exact algorithms are usually not capable of solving large-scale instances to optimality. Consequently, researchers have to resort to heuristics to search for near optimal solutions. Among these algorithms, local search and local search-based (meta)heuristics comprise the majority of these promising categories of algorithms (Rosing et al. 1999; Resende and Werneck 2004; Ren et al. 2012a). In these algorithms, various local search algorithms play a crucial role to improve the solution quality. Typical examples of local search algorithms for the p-median problem include interchange, 2-opt, and LK-search. These three local search algorithms share the commonality that, they all consider swap-based neighborhood structures, i.e., the neighborhood move operations are conducted by swapping the median variables and the non-median variables of the incumbent solution. Besides, each local search has its unique features, which are described as follows.

Interchange is proposed in the context of the p-median problem in (Teitz and Bart 1968), which is based on iteratively swapping one pair of variables, in the hope of improving the solution quality. Once such swap is not possible, interchange would terminate, and return the corresponding local optimum. Since the proposal, interchange has become the most widely used local search algorithm in the p-median literature. We shall also note that, there exist several popular implementations of interchange (Teitz and Bart 1968; Whitaker 1983; Resende and Werneck 2003), among which the version proposed by Resende and Werneck (2003) is the most efficient one, which benefits from a well designed time-space trade-off.

2-opt for the p-median problem is a local search algorithm with a larger neighborhood than interchange (Rosing et al. 1999). Each 2-opt neighborhood consists of the solutions that are reachable from the incumbent solution by swapping at most two pairs of variables. Larger neighborhood structure usually yields better solution quality. However, due to the neighborhood traversing overhead, 2-opt is usually not applicable to large-scale instances.

LK-search is a relatively new local search to solve the p-median problem, compared to other local search algorithms. Proposed by Kochetov et al. (2005), LK-search is among the best local search algorithms for the p-median problem. LK-search is based on the ideas of (Kernighan and Lin 1970) for the Graph Partition Problem (GPP). Unlike other swap-based local search (such as interchange and 2-opt) in which only swaps with improvement could be conducted, LK-search intends to overcome the local optimum traps by allowing the swaps that lead to worse solutions. Besides, LK-search also benefits from a tabu-like (Glover 1989, 1990) mechanism. Each time a swap operation is performed, the variables involved in the swap are put into tabu lists, to help the search explore new regions of the solution space. Besides, we should

note that, in the worst case, LK-search requires exponential number of swaps before convergence (Alekseeva et al. 2008).

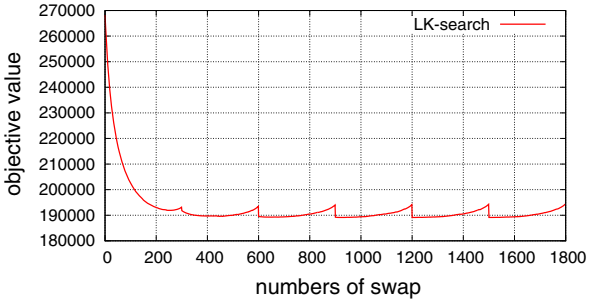
Algorithm 1: LK-search

```
   Input: Initial solution s
   Output: Improved solution s*
1  begin
2      flag ← true
3      s* ← s
4      while flag do
5          flag ← false
6          incumbent ← s*
7          candidate_in ←{facility i: i is not a median in incumbent}
8          candidate_out ←{facility j: j is a median in incumbent}
9          while candidate_in ≠ ∅ and candidate_out ≠ ∅ do
10             (in, out) ← findBestNeighbor(incumbent, candidate_in, candidate_out)
11             swap(incumbent, in, out)
12             candidate_in ← candidate_in \ {in}
13             candidate_out ← candidate_out \ {out}
14             if incumbent is better than s* then
15                 s* ← incumbent
16                 flag ← true
17             end
18         end
19     end
20     return s*
21 end
```

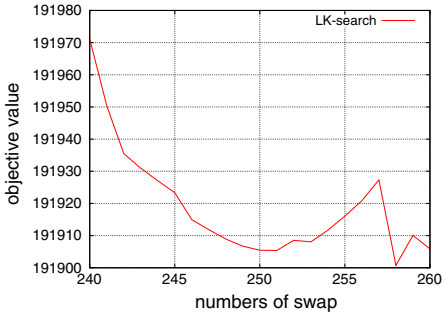
The pseudo code of LK-search is presented in Algorithm 1. The algorithm works as follows. Similar with most existing local search algorithms for solving the p-median problem, LK-search follows the iterative paradigm to improve the input solution s (Lines 4–19).[2] In the algorithm, Lines 9–18 represent the process of traversing each LK neighborhood. To be more specific, the incumbent solution is assigned with the currently best solution achieved up the previous iteration s^* (Line 6), and the candidate sets (candidate$_{out}$ and candidate$_{in}$) are firstly constructed, with the medians and the non-median facilities, respectively (Lines 7–8). Then, a series of swaps are performed, between the variables from the two candidate sets to improve the solution quality. Each swap involves the pair of variables that yields the best profit. [3] Each time a pair of variables is selected, the swap is conducted (Line 11). As mentioned, the main feature of LK-search, which differs from other swap-based local search, is that LK-search introduces a tabu-list like mechanism. As shown in Lines 12–13, each time a pair of

[2] In this study, we are mostly interested in the local search procedure, hence, for the constructive heuristic, we simply employ the random initialization (i.e., p out of m facilities are selected as medians uniformly at random).

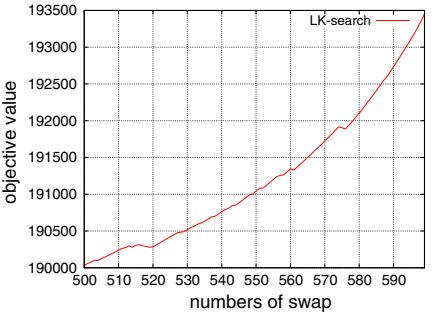
[3] The functionality of the subroutine **findBestNeighbor** in Line 10 is to find the best pair of variables that leads to the best profit from the candidate sets. In this study, this subroutine is partially based on the implementation of Resende and Werneck (2003).



(a) LK-search



(b) LK-search, partial enlarged view



(c) LK-search, partial enlarged view

Fig. 1 A typical run of LK-search, over PCB3038 ($p = 300$)

variables are swapped, these variables are excluded from the candidate sets.[4] With this mechanism, LK-search can to some extent overcome the traps of plateaus and ridges. After the current neighborhood has been traversed, and the best solution is updated, the search will continue, from the best solution just reached. Otherwise, the local search would terminate, and the best solution s^* is returned.

Figure 1a illustrates a typical execution of LK-search, over a random initial solution, to the benchmark instance PCB3038 with $p = 300$. The axes indicate the number of swaps conducted, and the objective value of the incumbent solution. In order to better interpret the LK-search's behavior, we also present two partial enlarged views of Fig. 1a, in Fig. 1b, c. From the figures, we could observe the following phenomena.

First, over the instance, the typical LK-search run in Fig. 1 requires 1,800 number of swaps to terminate, which consists of 6 Lin-Kernighan neighborhood traversals. Meanwhile, if we execute LK-search from 100 random initial solutions, we observe that the number of swaps ranges within [1200, 2400], with an average of 1,646 number of swaps. Second, the major improvement happens during the beginning of the search, especially the first LK neighborhood traversal. After that, the improvement tends to be smaller. Third, as mentioned, with the tabu-like mechanism, LK-search is able to

[4] Note that the variant presented in this paper is slightly different from the original version by Kochetov et al. (2005), in that during the preliminary experiments, we observe that the version presented in Algorithm 1 is generally more effective.

overcome certain traps of local optima. For example, within swaps number 250–260 (see Fig. 1b), the objective value of the incumbent solution undergoes an increase first, and a decrease subsequently. Fourth, despite the promising tabu-like mechanism, there might be redundant swap operations, which do not lead to better solutions. For example, within swap number 500–599 (see Fig. 1c), the objective value of the incumbent solution keeps increasing, except for several minor decreases, and the objective value is always above 190,000.

In summary, with the tabu-like mechanism, LK-search is able to achieve high quality solutions. However, there is still room for improvement. For instance, in LK-search, the first neighborhood traversal is critical to the search performance. Thus, if we could improve the traversal procedure for this neighborhood, the overall performance might be significantly enhanced. Second, as we have discussed, there might be computation waste in the search procedure. If we could detect the non-promising search directions, and prevent the search from exploring these directions, the LK-search procedure may be accelerated.

2.2 Mathematical programming assisted heuristics

In this subsection, we introduce the related work in which relaxation-based techniques and heuristics approaches are combined for problem solving. For example, Puchinger and Raidl (2008) propose a Relaxation Guided Variable Neighborhood Search (RGVNS) for the Multidimensional Knapsack Problem (MKP), in which the exploration order of the neighborhoods is sorted with the estimated quality corresponding to each neighborhood, based on the lower bound calculation achieved by Lagrangian relaxation. Ren et al. (2012a) issue the search space reduction in the context of the p-median problem. The reduction is carried out by calculating the lower bound of a series of partial solutions, using Lagrangian relaxation. Each time the lower bound exceeds the upper bound of the instance, the value assignment of the corresponding variable could be determined. With the search space reduction mechanism, the proposed algorithm Accelerated Limit Crossing based Multilevel Algorithm (ALCMA) updates several best known upper bound in the literature. From these examples, we can see that the performances of heuristics could be significantly improved, under the guidance of the information that is extracted from the relaxed problems. However, the information exploitation in the existing work is mostly realized at the global level, such as sorting the neighborhood order (Puchinger and Raidl 2008) and search space reduction (Ren et al. 2012a). To the best of our knowledge, the issue of guiding the search at the local level with the relaxation information has not yet been systematically investigated. Hence, in this study, we intend to explore the possibility of utilizing the information obtained from the relaxed problem, to achieve the two objectives raised in Sect. 2.1. The main difference between this study and the existing work is that, we intend to exploit the information at a finer granularity, which might enable more flexible designs. For instance, we could embed such algorithms into other frameworks.

After briefly introducing the mathematical programming assisted heuristics, we shall briefly revisit the Lagrangian relaxation procedure of the p-median problem (Senne and Lorena 2000). More specifically, the p-median problem can be modeled

as the following formulations:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{m} d_{ij} x_{ij} \tag{1}$$

s.t.

$$\sum_{j=1}^{m} x_{ij} = 1, \quad i \in \{1, 2, \ldots, n\} \tag{2}$$

$$\sum_{j=1}^{m} y_j = p \tag{3}$$

$$x_{ij} \leq y_j, \quad i \in \{1, 2, \ldots, n\}, j \in \{1, 2, \ldots, m\} \tag{4}$$

$$x_{ij}, y_j \in \{0, 1\}, \quad i \in \{1, 2, \ldots, n\}, j \in \{1, 2, \ldots, m\} \tag{5}$$

In the formulations, a solution consists of a matrix $X_{n \times m}$, as well as a vector Y_m specifying the subset of facilities are chosen. For the solution, $x_{ij} = 1$ means that user i is assigned to facility j, and $x_{ij} = 0$ otherwise; $y_j = 1$ represents that facility j is chosen as a median, and $y_j = 0$ otherwise.

Lagrangian relaxation involves solving the relaxed version of the problem, to obtain the lower bound. Senne and Lorena (2000) propose an efficient way to calculate the lower bound. In this paper, the lower bound is calculated based on this method. Hence, we shall provide some detailed information about the method in Senne and Lorena (2000).

In Senne and Lorena (2000), the relaxed version of the p-median problem is formulated by removing Constraint (2), meanwhile introducing a penalty term (parameterized by t and λ) for violating those removed constraint. Consequently, the problem can be transfered into the relaxed version:

$$\begin{aligned} &\min \sum_{i=1}^{n} \sum_{j=1}^{m} d_{ij} x_{ij} + t \sum_{i=1}^{n} \lambda_i (1 - \sum_{j=1}^{m} x_{ij}) \\ &= \min \sum_{i=1}^{n} \sum_{j=1}^{m} (d_{ij} - t\lambda_i) x_{ij} + t \sum_{i=1}^{n} \lambda_i \end{aligned} \tag{6}$$

s.t. Constraints (3), (4), and (5).

Given t and λ, it is straightforward that the X variables only appear in the first term, which could be decomposed into n sub-problems:

$$\min \sum_{j=1}^{m} (d_{ij} - t\lambda_i) x_{ij} \tag{7}$$

s.t. Constraints (3), (4), and (5).

Each sub-problem could be solved as follows: for each facility j, let an auxiliary cost be defined as $\beta_j = \sum_{i=1}^{n} \min(0, d_{ij} - t\lambda_i)$. Sort β in ascending order, such that $\beta_{k_1} \leq \beta_{k_2} \leq \cdots \leq \beta_{k_m}$, and let $K = \{k_1, k_2, \ldots, k_p\}$ be the index set associated with the p smallest β elements. Variables of Y corresponding with K are determined to be 1, and the rest variables of Y are assigned to 0. With the values of vector Y decided, the variables of X are then calculated:

$$x_{ij} = \begin{cases} 1 & d_{ij} < t\lambda_i, y_j = 1 \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

For any configurations of t and λ, Senne and Lorena (2000) show that the lower bound of the problem (denoted as LB), could be calculated by:

$$LB = \sum_{j=1}^{m} \beta_j y_j + t \sum_{i=1}^{n} \lambda_i = \sum_{j=1}^{p} \beta_{k_j} + t \sum_{i=1}^{n} \lambda_i \tag{9}$$

Algorithm 2: LagrangianRelaxation

Input: p-median problem instance π
Output: Lower bound LB, β, K, t, λ

```
begin
    Initialize t and λ
    while Criteria not met do
        for j = 1 to m do
            β_j = Σ_{i=1}^{n} min(0, d_ij − tλ_i)
        Sort the β vector so that β_{k_1} ≤ β_{k_2} ≤ ... ≤ β_{k_m}
        Calculate the lower bound with Equation (9)
        if The lower bound is improved then
            Update LB
        Update t and λ with respect to Senne and Lorena (2000)
    return LB, β, K, t, λ
```

3 Lagrangian relaxation assisted neighborhood search

In this section, we present how to exploit the structure obtained from the relaxed problem to guide the LK-search procedure. The proposed algorithm LANS: Lagrangian relaxation Assisted Neighborhood Search improves LK-search from two aspects: (1) how to avoid the potentially low quality attraction basins, and (2) how to detect the redundant swap operations, and cut off the non-promising search procedure.

The main idea of this study is to exploit the information gathered from the relaxed version of the p-median problem, to guide the local search process. We intend to conduct a reduction over the candidate sets during the first neighborhood traversal of LK-search, in the hope of avoiding the low quality attraction basins. The motivation behind this reduction mechanism is that, fixing a variable as a median (non-median) in essence introduces an extra constraint to the problem. This operation may lead to

Algorithm 3: strengthenedLowerBound

Input: p-median problem instance π, fixed median set M, fixed non-median set T
Output: Lower bound $LB(M, T)$

```
1 begin
2     Obtain LB, β, K, t, λ from Algorithm 2
3     for each facility i ∈ M do
4         Add y_i = 1 as an extra constraint
5     for each facility j ∈ T do
6         Add y_i = 0 as an extra constraint
7     Solve the relaxed problem of Equation (6), with the extra constraints
8     LB(M, N) ← optimal objective value to the relaxed problem
9     return LB(M, T)
```

the increase of the lower bound. Inspired by the popular branching rules in the tree search-based algorithms, such as beam search (Croce et al. 2004; Bennell and Song 2010) and local branching (Fischetti and Lodi 2003), during the first neighborhood traversal, we eliminate those variables that may cause the increase of the lower bound, from the candidate sets. We define the candidate sets as

$$candidate_{in} = \{\text{facility } i : i \text{ is not a median}, \beta_i - \beta_{k_p} < \epsilon\} \tag{10}$$

$$candidate_{out} = \{\text{facility } j : j \text{ is a median}, \beta_{k_{p+1}} - \beta_j < \epsilon\} \tag{11}$$

The structures β and K are extracted from the Lagrangian relaxation algorithm, which is mentioned in Sect. 2.2. It is straightforward to verify the reduction criteria. For example, for any facility i for which $\beta_i > \beta_{k_p}$ holds, forcing i to be a median will increase the lower bound of the corresponding solutions. The reason is that, by introducing the constraint $y_i = 0$ in the Lagrangian relaxation algorithm, the relaxed problem transfer to a $p-1$ dimensional problem. Hence, the $p-1$ smallest β values are to be selected. With the penalty parameters t and λ unchanged, $LB - \beta_i + \beta_{k_p}$ is a lower bound of the solution space with the constraint $y_i = 0$, which implies that the lower bound would increase at least ϵ, which is a sufficiently small positive floating point number (1×10^{-4} in this study). Hence, by replacing the equations of Lines 7–8 of LK-search to Eqs. 10 and 11, we realize the neighborhood reduction mechanism.

Then, we shall proceed to address the redundancy detection. The idea is similar with the neighborhood reduction. Note that in LK-search, the tabu-like mechanism guarantees that the swapped variables are held unchanged for the rest of the neighborhood traversal, which is similar as the bounding operation of the classical branch-and-bound algorithms (Järvinen et al. 1972; Christofides and Beasley 1982). More specifically, each time a swap operation is carried out, the lower bound is to be strengthened. If the updated lower bound exceeds the upper bound, the following neighborhood traversal can not reach any optimal solutions. Hence, the neighborhood traversal can be safely terminated. More detailed, each time a pair of facilities (in, out) is swapped, the lower bound is updated with Algorithm 3. The motivation of **strengthenedLowerBound** is similar with Eqs. 10 and 11, which tries to strength the lower bound by introducing extra constraints, which has also been used in several backbone guided approaches (Climer and Zhang 2002; Ren et al. 2012a). By comparing the strengthened lower

bound with the current best solution quality each time a swap is carried out, the redundancy detection mechanism could be incorporated in LK-search.

Algorithm 4: LANS

Input: Initial solution s, Lower bound LB, Auxiliary structures β and K from Algorithm 2
Output: Improved solution s^*

```
1  begin
2      flag ← true
3      s* ← s
4      threshold ← ϵ
5      while flag do
6          flag ← false
7          incumbent ← s*
           // Neighborhood reduction mechanism
8          candidate_in ← {i: i is not a median in incumbent, β_i − β_{k_p} < threshold}
9          candidate_out ←{j: j is a median in incumbent, β_{k_{p+1}} − β_j < threshold}
10         M ← ∅
11         T ← ∅
12         while candidate_in ≠ ∅ and candidate_out ≠ ∅ do
13             (in, out) ← findBestNeighbor(incumbent, candidate_in, candidate_out)
14             swap(incumbent, in, out)
15             candidate_in ← candidate_in \ {in}
16             candidate_out ← candidate_out \ {out}
               // Redundancy detection mechanism
17             M ← M ∪ {in}
18             T ← T ∪ {out}
19             if strengthenedLowerBound(M, T) > objective value of s then
20                 break
21             end
22             if incumbent is better than s* then
23                 s* ← incumbent
24                 flag ← true
25             end
26         end
27         threshold ← +∞
28     end
29     return s*
30 end
```

Finally, after introducing the two mechanisms, we present the pseudo code of LANS in Algorithm 4. The flow of LANS is similar as LK-search. The main differences lie in the following three aspects. (1) In Lines 8–9, during the first neighborhood traversal, we restrict the candidates to be those variables that do not lead to increased lower bounds when involved in swap operations. (2) During the subsequent neighborhood traversals, we switch back to the regular Lin-Kernighan neighborhood, by setting the threshold to a sufficiently large value (Line 27). By this strategy, we intend to make the search procedure more exploratory. (3) In Lines 17–21, after each swap operation, we update the lower bound, and conduct the redundancy detection to avoid the potential computational waste.

In summary, in this section, we propose the Lagrangian relaxation Assisted Neighborhood Search. In the following section, we intend to conduct various experiments, to evaluate the proposed algorithm.

4 Empirical study

In this section, we shall present a series of experiments, to examine the performance of the proposed local search algorithm, from both the effectiveness and the efficiency perspectives. We first demonstrate the effectiveness of the proposed algorithm using the numerical results over a series of benchmark instances. Then, we examine the behavior of LANS, when embedded in other frameworks. Finally, we employ the run-time distribution to evaluate the dynamic characteristics of LANS. All the algorithms are implemented in C++, compiled with g++ 4.7 with flag –O3. The experiments are conducted on a Pentium IV 3.2 GHz PC with 4GB memory, running GNU/Linux with kernel 3.10.

4.1 Numerical results

In this section, the numerical results of LANS are presented. To evaluate the performance of LANS, we consider four classes of benchmark instances, which are widely adopted in the p-median related literatures (Hansen and Mladenovic 1997; Resende and Werneck 2003, 2004; Kochetov et al. 2005).

The first class consists of 40 graph-based instances, which are from ORLIB (Beasley 1993). Each ORLIB instance is represented by a connected graph and a corresponding p value. Each node in the graph indicates both a facility and a user, and the distances between the users and the facilities are determined by the length of the shortest path between the corresponding nodes in the graph.

The second class consists of the Euclidean instances, which are constructed using TSPLIB (Reinelt 1991). In each Euclidean instance, the points on the plane represent both the user set and the facility set. The distances are Euclidean distances between these points. More specifically, we consider FL1400, PCB3038, RL5934, and RL11849, with various values of p. Following the experimental designs of the existing work (Hansen and Mladenovic 1997; Resende and Werneck 2004), the distances within each instance are kept with double precision.

The third class is RW, which consists of the random instances, which are introduced in Resende and Werneck (2003). In each RW instance, the number of the users and the number of the facilities are the same, and the distances between the users and the facilities are randomly generated.

The fourth class comprises of the instances with large duality gaps (denoted as GAP). These instances are introduced by Kochetov et al. (2005), and are generated to examine the performance of LK-search. Unlike the other three classes of instances, the density of the GAP instances are relatively low, i.e., not all facilities and the users are connected. Consequently, given a GAP instance, a median set of size p may not always be feasible (Kochetov et al. 2005).

To objectively measure the solution quality achieved by LANS, two baseline results are employed in this study. First, we use the best known results extracted in the literature (Resende and Werneck 2004; Pullan 2008; Ren et al. 2012a, b, 2013). The reason we use these results is that, among the instances, only the ORLIB instances have been exactly solved (Beasley 1993). However, the scales of these instances are relatively

small (for the largest ORLIB instance pmed40, the number of feasible solutions is $\binom{900}{90}$). Meanwhile, the scales of the Euclidean instances are much larger. For example, for the largest instance (RL11849 with $p = 1000$), the number of feasible solutions is $\binom{11849}{1000}$, which makes it too hard for exact algorithms. As a compromise, we introduce the best known results, and use the average percentage error rate as the measurement, which is defined following (Hansen and Mladenovic 1997):

$$\%\ err = \frac{C_{avg} - C_{opt}}{C_{opt}} \times 100, \tag{12}$$

where C_{opt} and C_{avg} indicate the best known upper bound and the average objective value of solutions achieved from multiple runs.

Second, LK-search is also used to validate the effect of the proposed mechanisms in this paper, in that LANS intends to improve the performance of LK-search by exploiting the structure of the relaxed problem. By comparing the results of LANS and LK-search, we hope to investigate whether the proposed mechanisms work. To be more specific, the comparisons are conducted as follows. For each benchmark instance, 100 random initial solutions are generated. Over these random solutions, LK-search and LANS are executed to improve the quality, respectively. Note that for the GAP instances, since local search may get trapped in non-feasible solutions (Kochetov et al. 2005), we shall repeat the executions of LANS, until 100 feasible output solutions are obtained. The numerical results are presented in Tables 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 and 12. Each table is organized as follows. The first column indicates the instances. The second column presents the best known objective value for each instance. Then, in columns 3–6 and columns 7–10, the results of LK-search and LANS are presented, respectively. For each algorithm, we give the minimum objective value achieved (min), the average percentage error rate ($\%\ err$), the standard deviation (SD), and the average running time measured in seconds (time). Finally, in column 11, we list the execution time of the Lagrangian relaxation pre-processing (measured in seconds), which is required for LANS. For each instance, the better objective value between the two algorithms is highlighted in boldface, and the shorter execution time is underlined. From the numerical results, several interesting observations could be drawn.

First, when we compare the best results achieved by LANS and LK-search, we can find that LANS outperforms LK-search. For all the instances, LANS is able to obtain solutions with objective value no worse than LK-search over 215 instances. When we consider the average quality of the two algorithms (measured by $\%\ err$), similar phenomena could be observed. The $\%\ err$ of LANS is better than or equal to LK-search over 173 out of 215 benchmark instances.

Among the four classes of instances, LANS performs well over the ORLIB, the Euclidean, and the RW instances, yet performs similarly as LK-search over the GAP instances. The reason might be that, the GAP instances have larger duality gaps. In this case, LANS might degenerate into LK-search, since LANS relies on the lower bound to realize both the neighborhood reduction and the redundency detection mechanisms.

To confirm this, in Fig. 2 we illustrate the $\%\ err$ differences between LANS and LK-search, against the duality gaps of the benchmark instances. Each sub-figure corre-

Table 1 Numerical results over ORLIB instances

Id	Best	LK-search				LANS				
		Min	*% err*	SD	Time	Min	*% err*	SD	Time	LR
1	5,819	**5,819**	0.00	0.00	< 0.01	**5,819**	0.00	0.00	< 0.01	0.01
2	4,093	**4,093**	0.17	5.76	< 0.01	**4,093**	0.18	5.82	< 0.01	0.03
3	4,250	**4,250**	0.00	0.00	< 0.01	**4,250**	0.00	0.00	< 0.01	0.02
4	3,034	**3,034**	0.15	5.36	< 0.01	**3,034**	0.03	2.55	< 0.01	0.02
5	1,355	**1,355**	0.07	1.41	< 0.01	**1,355**	0.01	0.59	< 0.01	0.02
6	7,824	**7,824**	0.00	0.00	0.02	**7,824**	0.00	0.00	0.02	0.06
7	5,631	**5,631**	0.00	0.00	0.01	**5,631**	0.00	0.00	0.01	0.05
8	4,445	**4,445**	0.00	0.00	0.01	**4,445**	0.00	0.40	< 0.01	0.04
9	2,734	**2,734**	0.42	6.91	< 0.01	**2,734**	0.22	6.35	< 0.01	0.02
10	1,255	**1,255**	0.24	3.85	0.01	**1255**	0.05	1.73	< 0.01	0.04
11	7,696	**7,696**	0.00	0.00	0.04	**7,696**	0.00	0.00	0.04	0.11
12	6,634	**6,634**	0.00	0.00	0.03	**6,634**	0.00	0.00	0.02	0.10
13	4,374	**4,374**	0.06	4.22	0.01	**4,374**	0.03	3.46	< 0.01	0.05
14	2,968	**2,968**	0.05	1.79	0.01	**2,968**	0.02	1.45	< 0.01	0.04
15	1,729	1,730	0.29	3.40	0.01	**1,729**	0.13	2.47	< 0.01	0.05
16	8,162	**8,162**	0.00	0.00	0.14	**8,162**	0.00	0.00	0.14	0.27
17	6,999	**6,999**	0.03	2.00	0.06	**6,999**	0.03	2.00	0.06	0.16
18	4,809	**4,809**	0.07	2.03	0.02	**4,809**	0.04	1.94	0.01	0.09
19	2,845	**2,845**	0.16	3.07	0.01	**2,845**	0.04	1.55	0.01	0.06
20	1,789	**1,789**	0.06	1.59	0.02	**1,789**	0.05	1.44	0.01	0.06
21	9,138	**9,138**	0.00	0.00	0.19	**9,138**	0.00	0.00	0.14	0.37
22	8,579	**8,579**	0.17	32.79	0.15	**8,579**	0.15	31.04	0.14	0.28
23	4,619	**4,619**	0.05	3.81	0.02	**4,619**	0.02	2.55	0.01	0.08
24	2,961	**2,961**	0.13	3.11	0.02	**2,961**	0.06	2.17	0.01	0.08
25	1,828	**1,828**	0.27	3.70	0.03	**1,828**	0.07	1.92	0.01	0.09
26	9,917	**9,917**	0.02	3.09	0.41	**9,917**	0.02	3.12	0.41	0.73
27	8,307	**8,307**	0.00	0.59	0.28	**8,307**	0.00	0.59	0.26	0.44
28	4,498	**4,498**	0.11	3.42	0.03	**4,498**	0.07	3.23	0.02	0.12
29	3,033	**3,033**	0.20	4.25	0.03	**3,033**	0.08	2.34	0.01	0.11
30	1,989	**1,989**	0.26	2.72	0.05	**1,989**	0.10	1.91	0.02	0.12
31	10,086	**10,086**	0.00	0.39	0.70	**10,086**	0.00	0.39	0.70	1.54
32	9,297	**9,297**	0.00	0.00	0.44	**9,297**	0.00	0.00	0.43	0.72
33	4,700	**4,700**	0.12	5.62	0.04	**4,700**	0.04	3.12	0.02	0.13
34	3,013	**3,013**	0.18	3.66	0.04	**3,013**	0.10	2.95	0.02	0.12
35	10,400	**10,400**	0.00	0.00	1.07	**10,400**	0.00	0.00	1.06	2.33
36	9,934	**9,934**	0.06	12.24	0.82	**9,934**	0.06	12.24	0.82	1.49
37	5,057	**5,057**	0.09	3.56	0.06	**5,057**	0.04	2.60	0.03	0.19
38	11,060	**11,060**	0.04	12.94	2.12	**11,060**	0.04	12.94	2.12	4.11
39	9,423	**9,423**	0.00	0.00	1.05	**9,423**	0.00	0.00	1.05	3.01
40	5,128	**5,128**	0.09	3.63	0.07	**5,128**	0.06	2.76	0.04	0.21

Table 2 Numerical results over FL1400 instances

p	Best	LK-search				LANS				
		Min	$\%\ err$	SD	Time	Min	$\%\ err$	SD	Time	LR
100	16,551.20	16,565.55	0.50	39.50	0.15	**16,557.75**	0.31	27.40	0.10	0.60
150	12,026.41	12,057.30	0.60	17.62	0.17	**12,048.39**	0.45	16.15	0.13	0.45
200	9,356.34	9,366.44	0.48	14.97	0.16	**9,361.19**	0.35	11.16	0.11	0.39
250	7,737.72	7,752.56	0.38	8.22	0.20	**7,751.45**	0.36	7.09	0.15	0.41
300	6,611.60	6,638.23	0.62	7.33	0.22	**6,634.20**	0.51	6.49	0.16	0.48
350	5,719.03	5,750.08	0.83	7.58	0.26	**5,740.82**	0.63	6.94	0.16	0.45
400	5,006.75	5,037.67	1.01	8.89	0.30	**5,024.31**	0.77	8.73	0.16	0.48
450	4,468.29	4,484.98	0.53	3.43	0.42	**4,480.02**	0.51	3.72	0.23	0.60
500	4,046.39	4,053.11	0.39	3.54	0.45	**4,051.78**	0.34	3.01	0.32	0.63

Table 3 Numerical results over PCB3038 instances

p	Best	LK-search				LANS				
		Min	$\%\ err$	SD	Time	Min	$\%\ err$	SD	Time	LR
100	352,609.60	353,527.96	0.84	833.06	1.10	**353,361.93**	0.59	665.73	1.01	4.24
150	281,163.12	282,123.93	0.75	556.62	0.75	**281,490.13**	0.49	427.48	0.66	3.26
200	238,344.20	239,108.03	0.65	340.53	0.71	**238,686.53**	0.48	304.87	0.54	2.68
250	209,206.90	209,748.75	0.51	232.71	0.74	**209,565.26**	0.36	209.65	0.53	2.38
300	187,689.40	188,259.13	0.55	225.86	0.77	**187,973.38**	0.37	163.63	0.50	2.16
350	170,919.46	171,481.58	0.61	196.77	0.91	**171,287.49**	0.38	147.01	0.51	2.13
400	157,027.21	157,587.30	0.65	191.81	0.95	**157,309.29**	0.39	147.33	0.53	2.06
450	145,362.91	145,948.89	0.63	161.02	1.00	**145,603.55**	0.39	135.47	0.51	2.07
500	135,447.39	136,073.02	0.70	138.50	1.06	**135,712.70**	0.40	127.39	0.55	3.03
550	126,825.24	127,431.50	0.74	140.16	1.12	**127,099.43**	0.43	102.60	0.56	1.97
600	119,059.77	119,747.76	0.80	131.89	1.21	**119,400.96**	0.45	103.91	0.61	3.04
650	112,017.65	112,719.35	0.88	125.02	1.32	**112,283.04**	0.49	100.30	0.71	3.16
700	105,823.96	106,423.74	0.81	121.84	1.51	**105,977.03**	0.47	117.13	0.69	2.24
750	100,331.25	100,824.37	0.77	116.25	1.49	**100,572.53**	0.44	87.24	0.72	2.20
800	95,374.17	95,840.01	0.72	99.59	1.52	**95,592.91**	0.38	64.90	0.67	2.18
850	90,983.64	91,411.64	0.69	87.36	1.71	**91,161.77**	0.38	58.53	0.69	2.41
900	86,972.78	87,404.14	0.70	88.85	1.77	**87,160.74**	0.36	63.67	0.74	2.36
950	83,267.38	83,596.69	0.63	77.64	1.75	**83,399.38**	0.32	52.94	0.78	2.38
1000	79,842.01	80,118.71	0.54	69.30	1.79	**79,948.56**	0.24	39.94	0.76	2.30

sponds to a class of instances, which is organized as follows. The x axis represents the relative duality gap ($\frac{C_{opt}-LB}{C_{opt}}$), and the y axis indicates the $\%\ err$ differences between LANS and LK-search. In each sub-figure, a point (x, y) indicates that there exists an instance with a relative duality gap x, and the difference between the $\%\ err$s of LANS and LK-search equals y ($\%\ err$ of LANS $-\%\ err$ of LK-search = y). Hence, the points below the reference line ($y = 0$) indicates those instances over which LANS obtains

Table 4 Numerical results over RL5934 instances

p	Best	LK-search				LANS				
		Min	% *err*	SD	Time	Min	% *err*	SD	Time	LR
100	2,724,817.16	2,729,656.92	0.67	5,772.65	7.70	**2729302.03**	0.47	4784.59	7.75	23.34
150	2,147,756.39	2,156,499.26	0.79	4,684.61	4.71	**2,151,399.47**	0.53	3,516.48	4.54	16.35
200	1,807,411.06	1,814,421.88	0.74	2,919.59	3.81	**1,811,137.79**	0.51	2206.32	3.48	13.70
250	1,569,712.98	1,576,071.17	0.69	2,291.88	3.80	**1,572,858.79**	0.45	1903.36	3.40	12.54
300	1,393,951.74	1,399,685.39	0.73	2,152.44	4.18	**1,397,012.47**	0.46	1653.64	3.69	12.72
350	1,256,599.00	1,261,506.21	0.66	1,684.65	4.04	**1,258,918.15**	0.42	1144.51	3.62	11.76
400	1,145,271.28	1,149,920.37	0.67	1,246.40	4.45	**1,147,892.71**	0.43	1141.91	3.73	11.70
450	1,053,024.59	1,057,308.15	0.63	1,174.31	4.84	**1,055,177.54**	0.38	1,052.08	3.65	11.09
500	973,920.22	977,192.98	0.63	1,079.02	4.91	**975,737.05**	0.36	800.65	3.44	11.65
600	848,217.37	850,431.49	0.45	760.40	5.24	**849,413.86**	0.26	507.18	3.21	11.50
700	751,972.31	753,975.64	0.45	666.24	5.67	**753,088.87**	0.25	442.53	3.21	11.48
800	676,724.24	678,580.69	0.42	435.83	6.37	**677,322.92**	0.22	340.46	3.42	12.09
900	613,308.88	615,191.38	0.44	388.80	6.69	**613,857.55**	0.23	344.43	3.47	11.84
1000	558,783.56	560,581.26	0.44	394.31	7.20	**559,364.27**	0.21	319.07	3.71	12.23
1100	511,778.61	513,131.23	0.44	355.85	7.98	**512,297.86**	0.21	262.00	3.75	12.73
1200	470,264.28	471,625.72	0.43	335.35	8.95	**470,705.96**	0.19	219.08	3.79	13.63
1300	433,543.91	434,572.67	0.47	330.07	9.72	**433,877.93**	0.20	178.53	4.11	14.23
1400	401,820.03	403,059.87	0.43	262.21	9.60	**402,084.12**	0.15	167.16	4.03	13.99
1500	373,987.02	375,036.14	0.41	232.85	11.04	**374,172.32**	0.15	163.71	4.39	15.34

Table 5 Numerical results over RL11849 instances

p	Best	LK-search				LANS				
		Min	% *err*	SD	Time	Min	% *err*	SD	Time	LR
100	5,850,644	**5,860,966**	0.65	14,401.81	126.39	5,864,272	0.56	10,155.18	111.13	192.63
200	4,003,810	4,019,337	0.79	5,893.12	27.44	**4,014,161**	0.58	5,338.73	26.80	68.74
300	3,194,525	3,211,764	0.76	3,736.71	21.63	**3,202,958**	0.57	3,729.65	20.08	51.54
400	2,696,221	2,711,079	0.76	2,880.46	22.21	**2,705,049**	0.51	2,282.23	20.79	48.96
500	2,357,273	2,368,283	0.75	3,064.31	23.16	**2,364,045**	0.50	2,419.94	20.72	46.68
600	2,112,870	2,122,920	0.70	2,106.15	24.93	**2,119,268**	0.47	1,553.08	22.45	46.04
700	1,922,903	1,931,007	0.68	1,901.52	27.87	**1,927,946**	0.46	1,366.48	23.50	47.37
800	1,766,512	1,774,600	0.71	1,760.77	29.68	**1,771,267**	0.45	1,225.49	24.18	48.25
900	1,636,053	1,643,916	0.62	1,367.18	34.77	**1,640,117**	0.39	1,038.06	25.29	52.99
1000	1,522,955	1,528,409	0.57	1,066.35	36.89	**1,526,331**	0.35	725.36	25.66	44.27

better solutions, since over these instances, LANS is able to achieve smaller % *err*s. From Fig. 2, we could observe that the performance of LANS relies on the quality of the lower bound. For example, over the ORLIB, Euclidean, and the RW instances (Fig. 2a–c), the majority of the points lie below the reference line. Meanwhile, over these three classes of instances, the relative duality gaps tend to be small. In contrast,

Table 6 Numerical results over RW100 instances

p	Best	LK-search				LANS				LR
		Min	% *err*	SD	Time	Min	% *err*	SD	Time	
10	530.00	**530.00**	2.04	8.05	0.01	**530.00**	1.99	8.59	< 0.01	0.03
20	277.00	**277.00**	1.24	4.27	< 0.01	**277.00**	0.94	4.09	< 0.01	0.01
30	213.00	**213.00**	0.52	1.23	< 0.01	**213.00**	0.44	1.13	< 0.01	0.01
40	187.00	**187.00**	0.09	0.38	< 0.01	**187.00**	0.23	0.57	< 0.01	0.01
50	172.00	**172.00**	0.15	0.48	< 0.01	**172.00**	0.12	0.41	< 0.01	0.01

Table 7 Numerical results over RW250 instances

p	Best	LK-search				LANS				LR
		Min	% *err*	SD	Time	Min	% *err*	SD	Time	
10	3,691.00	**3,691.00**	2.23	74.15	0.08	**3,691.00**	2.32	78.06	0.08	0.14
25	1,360.00	**1,360.00**	2.98	23.12	0.03	1,364.00	3.39	25.39	0.03	0.06
50	713.00	**713.00**	1.63	5.01	0.01	**713.00**	1.54	5.70	0.01	0.04
75	523.00	**523.00**	0.78	3.05	0.01	**523.00**	0.44	1.99	< 0.01	0.03
100	444.00	**444.00**	0.12	0.66	0.01	**444.00**	0.17	0.75	< 0.01	0.03
125	411.00	**411.00**	0.06	0.49	0.01	**411.00**	0.08	0.51	< 0.01	0.03

Table 8 Numerical results over RW500 instances

p	Best	LK-search				LANS				LR
		Min	% *err*	SD	Time	Min	% *err*	SD	Time	
10	16,108.00	**16,135.00**	2.50	257.47	0.91	**16,135.00**	2.47	244.87	0.91	1.15
25	5,626.00	5,723.00	4.09	78.28	0.35	**5,716.00**	4.35	74.92	0.35	0.45
50	2,626.00	**2,631.00**	2.42	32.99	0.10	2,632.00	2.29	30.86	0.10	0.16
75	1,757.00	**1,757.00**	1.81	17.05	0.06	1759.00	1.64	16.03	0.05	0.12
100	1,379.00	**1,381.00**	1.13	7.45	0.04	1,382.00	1.05	6.65	0.04	0.10
150	1,024.00	1,025.00	0.63	3.57	0.03	**1,024.00**	0.36	2.32	0.01	0.08
200	893.00	**893.00**	0.20	1.38	0.03	**893.00**	0.11	0.89	0.01	0.07
250	833.00	**833.00**	0.11	0.76	0.03	**833.00**	0.03	0.50	0.01	0.07

over the GAP instances (Fig. 2d), the relative duality gaps are much larger. Consequently, the proportion of the points lying above the reference line is higher than those in Fig. 2a–c. This phenomenon to some extent explains why LANS performs similarly as LK-search over the GAP instances.

To obtain a more intuitive impression, we visually present the % *err* comparison between LANS and LK-search in Fig. 3a. In the figure, the x axis and the y axis indicate the % *err* obtained by LANS and LK-search, respectively. The comparison over each instance is represented as a point (x, y), which means that over the corresponding instance, the % *err*s of LANS and LK-search are x % and y %, respectively. We also plot the reference line $y = x$ in the figure. Hence, the points above the reference line indicate the benchmark instances over which LANS outperforms LK-search. From the

Table 9 Numerical results over RW1000 instances

p	Best	LK-search				LANS				
		Min	$\%\ err$	SD	Time	Min	$\%\ err$	SD	Time	LR
10	67,811.00	**67,811.00**	2.75	1,144.31	14.89	**67,811.00**	2.71	1142.57	14.93	18.17
25	24,896.00	24,997.00	3.04	342.71	8.34	**24,896.00**	2.72	324.05	8.41	9.57
50	11,259.00	**11,301.00**	3.29	168.99	1.56	11,332.00	3.08	151.48	1.46	2.01
75	7,134.00	7,199.00	3.11	71.27	0.55	**7,179.00**	3.24	76.09	0.54	0.75
100	5,210.00	5,238.00	2.60	56.11	0.34	**5,235.00**	2.74	57.98	0.33	0.49
200	2,704.00	**2,714.00**	1.25	12.56	0.18	**2,714.00**	1.15	9.75	0.16	0.31
300	2,018.00	**2,018.00**	0.28	3.21	0.16	**2,018.00**	0.21	2.64	0.05	0.26
400	1,734.00	**1,734.00**	0.12	1.27	0.15	**1,734.00**	0.07	1.13	0.04	0.24
500	1,614.00	**1,614.00**	0.18	1.85	0.13	**1,614.00**	0.06	1.16	0.04	0.22

figure, it is obvious that LANS is able to achieve better $\%\ err$ over the majority of the instances. To confirm this, we employ the nonparametric Wilcoxon's signed rank test to check the potential differences in performance between the two algorithms. We set the null hypothesis to be that both algorithms in comparison have similar performance, and consider the 95 % confidence level. The result returned by the hypothesis test shows that LANS statistically outperforms LK-search, in terms of average percentage error rate (p value $< 1 \times 10^{-4}$).

Then we proceed to compare the execution time of the two algorithms. Similar with Fig. 3a, we illustrate the time comparison between LANS and LK-search in Fig. 3b. In the figure, the comparison results are organized in the same way as Fig. 3a, except that the axes represent the average running time of the two algorithms. From the figure, we could see that LANS is faster than LK-search. With the Wilcoxon's test, we state that LANS is faster than LK-search, at 95 % confidence level (p value $< 1 \times 10^{-4}$). Meanwhile, we should note that LANS requires that the Lagrangian relaxation procedure is executed as a pre-processing step. We argue that despite the extra computational overhead, the Lagrangian relaxation execution is worthy, based on the following reasons. On the one hand, in LANS, the pre-processing is required only once. On the other hand, local search such as LANS and LK-search is usually embedded in other algorithms. In these scenarios, local search would be repeatedly applied. Consequently, the computational overhead introduced by the pre-processing is amortized over the multiple executions of local search.

4.2 LANS as an embedded subroutine

In Sect. 4.1, the performance of LANS has been demonstrated to be effective, when executed as a stand-alone algorithm. However, it is more often that these local search algorithms be embedded into other heuristics. Hence, in this subsection, we intend to evaluate the performance of LANS as an embedded subroutine.

We employ the Euclidean (FL1400, PCB3038, and RL5934), the RW, and the GAP instances to conduct the experiment. We do not consider the other instances for the

Table 10 Numerical results over GAP-A instances

Instance	Best	LK-search				LANS				
		Min	% *err*	SD	Time	Min	% *err*	SD	Time	LR
1,032PM	127.00	**127.00**	9.08	8.57	< 0.01	**127.00**	8.24	7.60	< 0.01	0.01
1,132PM	163.00	**163.00**	6.84	7.99	< 0.01	**163.00**	7.85	7.86	< 0.01	0.01
1,232PM	123.00	**123.00**	14.36	12.12	< 0.01	**123.00**	14.07	11.44	< 0.01	0.01
1,332PM	164.00	**164.00**	1.74	2.92	< 0.01	**164.00**	1.21	2.50	< 0.01	0.01
1,432PM	150.00	**150.00**	4.38	5.56	< 0.01	**150.00**	3.99	5.43	< 0.01	0.01
1,532PM	158.00	**158.00**	0.00	0.00	< 0.01	**158.00**	0.00	0.00	< 0.01	0.01
1,632PM	141.00	**141.00**	1.76	3.78	< 0.01	**141.00**	2.10	4.09	< 0.01	0.01
1,732PM	157.00	**157.00**	0.74	0.99	< 0.01	**157.00**	0.74	0.99	< 0.01	0.01
1,832PM	135.00	**135.00**	7.80	6.92	< 0.01	**135.00**	7.93	6.84	< 0.01	0.02
1,932PM	146.00	**146.00**	2.72	6.61	< 0.01	**146.00**	2.13	5.48	< 0.01	0.02
2,032PM	150.00	**150.00**	8.73	7.47	< 0.01	**150.00**	9.52	7.66	< 0.01	0.01
2,132PM	140.00	**140.00**	4.38	6.02	< 0.01	**140.00**	4.06	5.39	< 0.01	0.01
2,232PM	145.00	**145.00**	0.00	0.00	< 0.01	**145.00**	0.00	0.00	< 0.01	0.02
2,332PM	172.00	**172.00**	1.12	2.45	< 0.01	**172.00**	1.05	2.41	< 0.01	0.01
2,432PM	137.00	**137.00**	9.05	8.54	< 0.01	**137.00**	8.66	7.43	< 0.01	0.01
2,532PM	153.00	**153.00**	2.04	6.48	< 0.01	**153.00**	1.31	5.45	< 0.01	0.01
2,632PM	164.00	**164.00**	4.35	4.91	< 0.01	**164.00**	3.75	4.39	< 0.01	0.01
2,732PM	123.00	**123.00**	18.31	14.33	< 0.01	**123.00**	17.68	15.29	< 0.01	0.01
2,832PM	145.00	**145.00**	6.07	6.00	< 0.01	**145.00**	5.43	4.61	< 0.01	0.01
2,932PM	155.00	**155.00**	2.36	7.74	< 0.01	**155.00**	2.25	7.07	< 0.01	0.02
3,032PM	113.00	**113.00**	9.66	10.39	< 0.01	**113.00**	7.79	8.99	< 0.01	0.02
3,132PM	130.00	**130.00**	11.32	10.84	< 0.01	**130.00**	9.98	11.19	< 0.01	0.01
3,232PM	157.00	**157.00**	10.91	4.60	< 0.01	**157.00**	10.38	4.36	< 0.01	0.01
332PM	154.00	**154.00**	4.23	6.00	< 0.01	**154.00**	4.21	6.52	< 0.01	0.01
432PM	155.00	**155.00**	1.12	4.24	< 0.01	**155.00**	1.32	4.53	< 0.01	0.01
532PM	150.00	**150.00**	4.33	4.80	< 0.01	**150.00**	3.62	5.02	< 0.01	0.01
632PM	162.00	**162.00**	1.96	4.32	< 0.01	**162.00**	1.85	3.83	< 0.01	0.01
732PM	157.00	**157.00**	0.00	0.00	< 0.01	**157.00**	0.00	0.00	< 0.01	0.01
832PM	136.00	**136.00**	5.78	10.08	< 0.01	**136.00**	4.54	9.26	< 0.01	0.02
932PM	133.00	**133.00**	7.24	10.30	< 0.01	**133.00**	5.89	9.74	< 0.01	0.02

following reasons. For the ORLIB instances, applying LANS over random initial solutions could achieve the optimality (see Sect. 4.1). Besides, the RL11849 instances are not incorporated due to their scales. As for the framework in which LANS is to be embedded, we adopt the ALCMA, due to its effectiveness. In the original version of ALCMA (Ren et al. 2012a), interchange is employed as the embedded local search subroutine. In this subsection, we replace interchange with LANS, and obtain the variant of ALCMA (denoted as ALCMA$_{LANS}$).

For each benchmark instance, we independently execute ALCMA$_{LANS}$ for 9 times, which follows the experimental design of the existing work (Hansen and Mladenovic

Table 11 Numerical results over GAP-B instances

Instance	Best	LK-search				LANS				
		Min	% *err*	SD	Time	Min	% *err*	SD	Time	LR
1,031PM	165.00	**165.00**	0.00	0.00	< 0.01	**165.00**	0.00	0.00	< 0.01	0.02
1,131PM	162.00	**162.00**	0.36	1.79	< 0.01	**162.00**	0.44	1.96	< 0.01	0.02
1,231PM	136.00	**136.00**	11.40	8.26	< 0.01	**136.00**	10.97	7.15	< 0.01	0.02
1,331PM	135.00	**135.00**	12.39	9.38	< 0.01	**135.00**	13.59	9.08	< 0.01	0.02
1,431PM	129.00	131.00	10.98	10.74	< 0.01	**129.00**	10.04	11.18	< 0.01	0.02
1,531PM	130.00	**130.00**	11.85	9.71	< 0.01	**130.00**	13.65	8.49	< 0.01	0.02
1,631PM	140.00	**140.00**	12.61	8.30	< 0.01	**140.00**	12.67	9.01	< 0.01	0.01
1,731PM	174.00	**174.00**	1.54	7.05	< 0.01	**174.00**	1.81	7.54	< 0.01	0.02
1,831PM	134.00	**134.00**	10.52	11.68	< 0.01	**134.00**	12.15	11.33	< 0.01	0.02
1,931PM	136.00	**136.00**	8.28	9.27	< 0.01	**136.00**	8.60	8.94	< 0.01	0.02
2,031PM	131.00	**131.00**	11.36	12.00	< 0.01	**131.00**	9.05	8.92	< 0.01	0.02
2,131PM	126.00	**126.00**	12.29	11.75	< 0.01	**126.00**	11.90	10.33	< 0.01	0.02
2,231PM	179.00	**179.00**	0.00	0.00	< 0.01	**179.00**	0.00	0.00	< 0.01	0.02
2,331PM	134.00	**134.00**	12.90	11.27	< 0.01	**134.00**	14.19	11.23	< 0.01	0.02
2,431PM	137.00	**137.00**	8.04	9.41	< 0.01	**137.00**	7.41	8.34	< 0.01	0.02
2,531PM	124.00	**124.00**	12.34	9.59	< 0.01	**124.00**	11.59	10.28	< 0.01	0.02
2,631PM	131.00	**131.00**	16.91	9.72	< 0.01	**131.00**	14.97	10.05	< 0.01	0.02
2,731PM	159.00	**159.00**	0.00	0.00	< 0.01	**159.00**	0.00	0.00	< 0.01	0.02
2,831PM	151.00	**151.00**	0.00	0.00	< 0.01	**151.00**	0.00	0.00	< 0.01	0.02
2,931PM	132.00	**132.00**	12.50	8.41	< 0.01	137.00	11.57	7.68	< 0.01	0.01
3,031PM	144.00	**144.00**	7.69	7.36	< 0.01	**144.00**	7.31	7.36	< 0.01	0.02
3,131PM	158.00	**158.00**	11.32	10.75	< 0.01	**158.00**	10.99	9.70	< 0.01	0.02
3,231PM	125.00	**125.00**	17.71	14.09	< 0.01	**125.00**	17.43	16.02	< 0.01	0.02
331PM	123.00	135.00	22.39	10.50	< 0.01	**123.00**	21.36	10.36	< 0.01	0.02
431PM	132.00	**132.00**	12.56	10.02	< 0.01	**132.00**	15.30	10.97	< 0.01	0.02
531PM	135.00	**135.00**	10.30	10.64	< 0.01	**135.00**	10.35	10.61	< 0.01	0.02
631PM	140.00	**140.00**	8.49	8.40	< 0.01	**140.00**	9.09	9.70	< 0.01	0.02
731PM	130.00	**130.00**	13.70	10.50	< 0.01	**130.00**	14.36	11.26	< 0.01	0.02
831PM	138.00	**138.00**	9.53	8.73	< 0.01	**138.00**	9.68	8.12	< 0.01	0.02
931PM	172.00	**172.00**	0.00	0.00	< 0.01	**172.00**	0.00	0.00	< 0.01	0.02

1997; Resende and Werneck 2004; Ren et al. 2012a). Over the 9 executions, we observe that ALCMA$_{LANS}$ is able to achieve very competitive performance over the Euclidean instances, yet is not quite effective over the RW instances and the GAP instances. Over the Euclidean instances, the % *err* of ALCMA$_{LANS}$ is always less than 0.09, while over the GAP instances, the maximum % *err* is larger than 10. In particular, ALCMA$_{LANS}$ updates the best known results in the literature, over 6 instances. Therefore, we present the numerical results over these instances in Table 13. [5] The table is

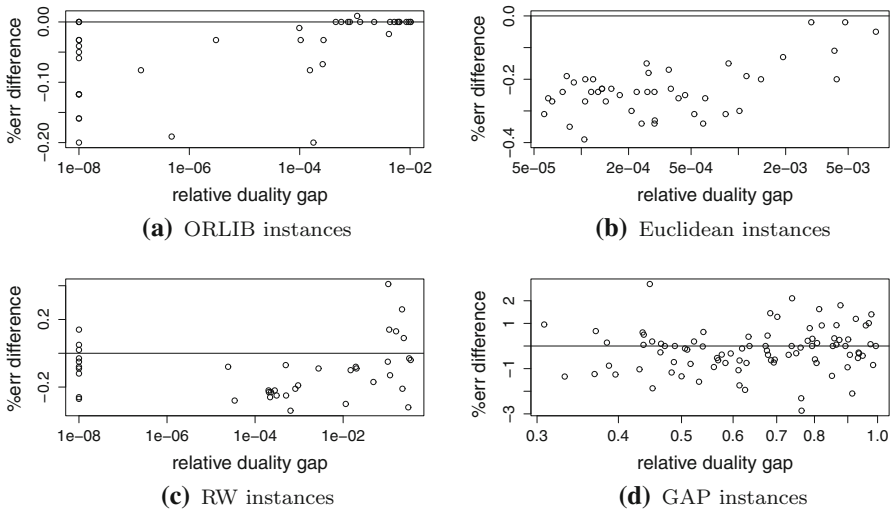
[5] More comprehensive numerical results could be found at http://oscar-lab.org/people/~zren/lans.

Table 12 Numerical results over GAP-C instances

Instance	Best	LK-search				LANS				
		Min	% *err*	SD	Time	Min	% *err*	SD	Time	LR
1,033PM	138.00	**138.00**	10.17	12.67	< 0.01	**138.00**	11.57	11.99	< 0.01	0.01
1,133PM	147.00	**147.00**	9.53	9.09	< 0.01	**147.00**	10.44	9.46	< 0.01	0.01
1,233PM	142.00	**142.00**	10.87	8.59	< 0.01	**142.00**	10.54	8.91	< 0.01	0.01
1,333PM	140.00	**140.00**	13.95	12.42	< 0.01	**140.00**	11.85	10.92	< 0.01	0.02
1,433PM	152.00	**152.00**	8.14	9.54	< 0.01	**152.00**	8.43	9.23	< 0.01	0.01
1,533PM	133.00	**133.00**	18.25	12.88	< 0.01	**133.00**	19.17	12.44	< 0.01	0.02
1,633PM	141.00	**141.00**	9.77	9.77	< 0.01	**141.00**	8.45	8.42	< 0.01	0.01
1,733PM	134.00	**134.00**	12.03	10.01	< 0.01	**134.00**	12.94	11.57	< 0.01	0.01
1,833PM	139.00	**139.00**	14.01	9.11	< 0.01	144.00	13.26	7.95	< 0.01	0.02
1,933PM	137.00	**137.00**	13.04	11.06	< 0.01	**137.00**	13.27	10.62	< 0.01	0.01
2,033PM	140.00	**140.00**	16.55	10.49	< 0.01	**140.00**	13.69	11.24	< 0.01	0.01
2,133PM	138.00	**138.00**	17.32	9.93	< 0.01	**138.00**	19.43	11.51	< 0.01	0.01
2,233PM	121.00	**121.00**	17.19	15.00	< 0.01	**121.00**	17.65	15.03	< 0.01	0.02
2,333PM	133.00	**133.00**	16.38	11.35	< 0.01	**133.00**	17.83	10.58	< 0.01	0.01
2,433PM	139.00	**139.00**	11.17	10.88	< 0.01	**139.00**	10.97	9.32	< 0.01	0.01
2,533PM	131.00	**131.00**	15.86	11.80	< 0.01	**131.00**	16.27	11.61	< 0.01	0.01
2,633PM	132.00	**132.00**	19.61	12.46	< 0.01	**132.00**	17.87	12.33	< 0.01	0.01
2,733PM	139.00	**139.00**	17.05	12.80	< 0.01	**139.00**	15.98	12.90	< 0.01	0.02
2,833PM	137.00	**137.00**	13.25	12.64	< 0.01	**137.00**	12.50	12.15	< 0.01	0.01
2,933PM	124.00	**124.00**	20.13	15.42	< 0.01	**124.00**	19.34	14.00	< 0.01	0.01
3,033PM	137.00	**137.00**	11.74	10.55	< 0.01	**137.00**	12.36	8.80	< 0.01	0.01
3,133PM	141.00	**141.00**	12.86	9.56	< 0.01	**141.00**	11.28	9.66	< 0.01	0.02
3,233PM	129.00	**129.00**	14.09	10.81	< 0.01	**129.00**	14.19	9.97	< 0.01	0.01
333PM	147.00	**147.00**	11.67	9.44	< 0.01	**147.00**	11.51	9.13	< 0.01	0.01
433PM	145.00	**145.00**	11.49	9.92	< 0.01	**145.00**	10.32	11.66	< 0.01	0.01
533PM	142.00	**142.00**	10.07	11.42	< 0.01	**142.00**	10.27	12.51	< 0.01	0.01
633PM	144.00	**144.00**	10.21	8.39	< 0.01	**144.00**	10.71	10.35	< 0.01	0.01
733PM	137.00	**137.00**	14.61	10.73	< 0.01	**137.00**	13.74	8.96	< 0.01	0.02
833PM	144.00	**144.00**	14.29	8.23	< 0.01	**144.00**	13.03	9.72	< 0.01	0.01
933PM	130.00	**130.00**	14.82	13.95	< 0.01	**130.00**	15.77	14.61	< 0.01	0.02

organized similarly as in Sect. 4.1, i.e, we present the minimum objective value, % *err*, SD, and average running time. For comparison, we also list the numerical results of ALCMA$_{LK}$, which is obtained by embedding LK-search into ALCMA.

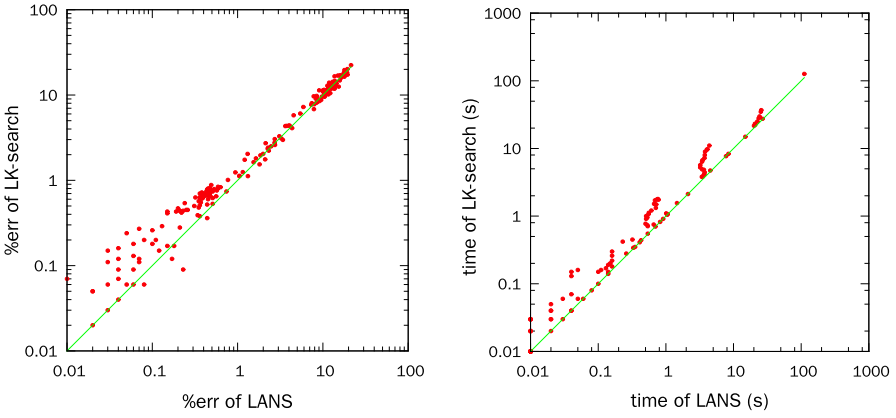
From Table 13, we could observe that, when embedded into ALCMA, the performance of LANS could be further improved, especially over the large Euclidean instances. Over the instances, ALCMA$_{LANS}$ dominates ALCMA$_{LK}$ in terms of both effectiveness and efficiency. From the effectiveness perspective, the best solutions achieved by ALCMA$_{LANS}$ update the best known results in the literature. From the efficiency perspective, ALCMA$_{LANS}$ requires much less time compared with



(a) ORLIB instances **(b)** Euclidean instances

(c) RW instances **(d)** GAP instances

Fig. 2 *% err* differences versus relative duality gap



(a) *%err* comparison **(b)** time comparison

Fig. 3 *% err* and time comparison, between LANS and LK-search

ALCMA$_{LK}$. These observations to some extent demonstrate the performance of LANS.

4.3 Run-time distribution analysis

In Sects. 4.1 and 4.2, we have demonstrated the effectiveness of LANS, especially over the large-scale Euclidean instances. To gain more insights about the dynamic characteristics of LANS and LK-search, the run-time distribution analysis is introduced. The run-time distribution, also known as time to target analysis (Hoos 2005; Aiex et al. 2007), is usually employed to capture various properties of heuristic algorithms, such

Table 13 Numerical results for ALCMA$_{LK}$ and ALCMA$_{LANS}$ over typical instances

Instances	p	Best	ALCMA$_{LK}$				ALCMA$_{LANS}$			
			Min	% *err*	SD	Time	Min	% *err*	SD	Time
FL1400	300	6,611.60	6,610.72	0.03	0.55	221.69	**6,610.71**	<0.01	0.88	117.30
PCB3038	300	187,689.40	**187,686.24**	<0.01	5.45	462.90	**187,686.24**	<0.01	5.09	230.75
PCB3038	600	119,059.77	119,059.77	<0.01	0.16	445.32	**119,058.09**	<0.01	0.91	144.49
PCB3038	700	105,823.96	105,823.97	<0.01	0.30	565.08	**105,822.49**	<0.01	0.78	228.58
RL5934	700	751,972.30	751,970.11	<0.01	3.27	3,376.85	**751,970.10**	<0.01	0.26	1,728.09
RL5934	1,100	511,778.60	511,778.61	<0.01	0.00	4,063.42	**511,764.80**	<0.01	4.61	2,017.19

as the convergence speed, the average solution quality, and so on. More specifically, the comparisons are carried out by running each algorithm over typical benchmark instances for multiple times, and examining the cumulative completions that the algorithm achieves a certain quality threshold as time elapses. In this subsection, we intend to investigate the reasons why LANS works, by examine the run-time behaviors of LANS and LK-search, as well as two other variant versions of LK-search.

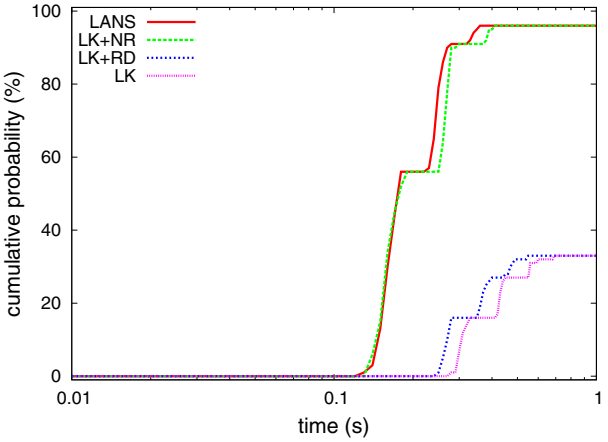
As for the typical benchmark instances, we employ PCB3038 ($p = 300$) and RL5934 ($p = 500$). We do not consider the other instances (such as the instances from ORLIB, RW, and GAP) because their scales are relatively small, and the local search algorithms terminate too fast. For the algorithms in comparison, besides LANS and LK-search, we consider two variants of LK-search, which combine LK-search with only one mechanism of neighborhood reduction and redundancy detection, respectively. Hence, these two variants are indicated as LK+NR and LK+RD. The purpose of introducing LK+NR and LK+RD in this experiment is to investigate the impact of the two mechanisms on LANS.

Over the instances, 100 random initial solutions are firstly generated, similarly as the experiment in Sect. 4.1. Then, over each random solution, each local search algorithm is executed independently. For each algorithm, its run-time behavior is represented by a run-time distribution curve determined from the 100 runs of the algorithm. In Fig. 4, the run-time distribution curve of each algorithm is illustrated as follows. The x axis indicates the log-scale time, and the y axis represents the cumulative probability that the algorithm achieves the predefined solution quality threshold. In this study, the threshold is set to be 0.05 % over the best known upper bound, because during the experiments, we find this threshold effective in distinguishing the performances of the algorithms.

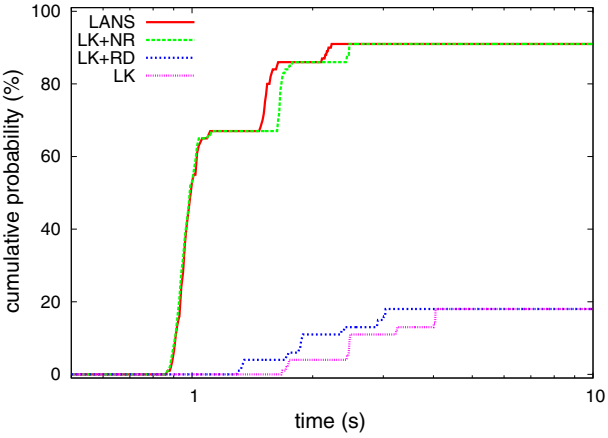
From the run-time distribution plots, we could draw the following observations.

First, similar with the numerical results, over both the instances, LANS shows a clear dominance over LK-search, in terms of both effectiveness and the efficiency. In Fig. 4a, b, the distribution curves of LANS lie above the curves of LK-search in comparison.

Second, when we compare LK+NR and LK-search, we can see that the neighborhood reduction mechanism significantly improves the solution quality. For example, in Fig. 4a, the distribution curve of LK+NR is alway above that of LK-search. Especially, when the algorithms terminate, the cumulative probability of LK+NR is over 90 %,



(a) PCB3038 ($p = 300$)



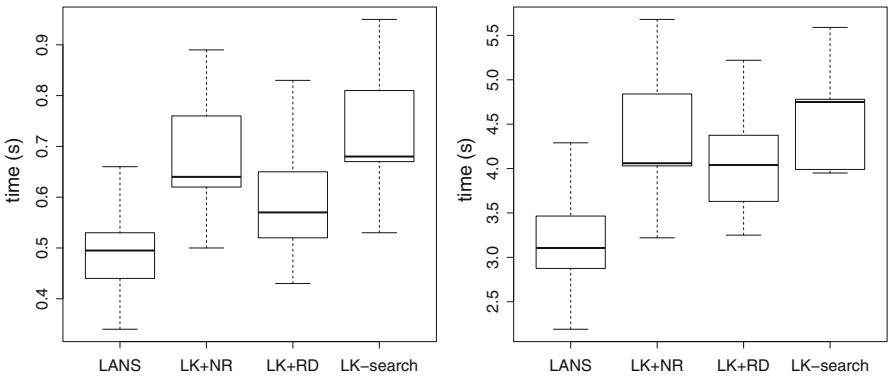
(b) RL5934 ($p = 500$)

Fig. 4 Run-time distribution plots for LANS, LK-search, and its variants

while the corresponding probability of LK-search is below 40 %. Similar phenomenon could be observed when we compare LANS and LK+RD, in that these two algorithms differ from each other only in the neighborhood reduction mechanism.

Third, comparing LK+RD and LK-search we observe that, with the redundancy detection mechanism, the distribution curve of LK+RD shifts to the left of LK-search. This phenomenon implies that, the redundancy detection mechanism is helpful in accelerating the local search procedure. Similar observations could be found when we compare LANS and LK+NR.

Interestingly, from Fig. 4, it seems that the effect caused by the redundancy detection mechanism is not as significant as that caused by the neighborhood reduction mechanism, since the distribution curves of LANS and LK+NR are close to each other. However, when we compare the execution time of the algorithms in Fig. 5, we could observe the influence of the redundancy detection mechanism. In terms of the



(a) PCB3038 ($p = 300$) **(b)** RL5934 ($p = 500$)

Fig. 5 Execution time comparison between LANS, LK-search, and its variants, using Box-Whisker plot

average execution time, LANS is much faster than LK+NR, and LK-search is slower than LK+RD. These observations could be drawn over both the benchmark instances, which to some extent demonstrate the usefulness of the redundancy detection mechanism.

In summary, with the run-time distribution analysis, we demonstrate that both the neighborhood reduction and the redundancy detection mechanisms are able to improve the performance of LK-search. By integrating these two mechanisms and LK-search, LANS is able to achieve competitive results.

5 Conclusion and future work

In this paper, we investigate the approaches of exploiting the information of the relaxed problem to guide the local search procedure. Using LK-search for the p-median problem as a case study, we propose the Lagrangian relaxation Assisted Neighborhood Search (LANS). The contributions of this paper could be summarized as follows.

- We propose a neighborhood reduction mechanism, to avoid the search from being trapped in low quality attraction basins.
- We develop a redundancy detection method, to detect and terminate the neighborhood traversal that could not find promising solutions.
- Extensive experiments demonstrate both the effectiveness and the efficiency of the proposed local search. Statistical tests show that LANS compares favorably to LK-search, which is among the state-of-the-art local search algorithms for the p-median problem.
- By embedding LANS into Accelerated Limit Crossing based Multilevel Algorithm (ALCMA), several best known results over benchmark instances could be updated.

For the future work, we would like to investigate the following issues. First, despite the effectiveness and the efficiency of LANS, we admit that the Lagrangian relaxation-based pre-process introduces extra computational overhead. The algorithm would be more efficient if the pre-processing could be further accelerated. Second, in the

numerical results, we observe that LANS performs well over most instances, but behaves similarly as LK-search over the GAP instances, due to the large duality gaps of these instances. In the future, we would like to investigate the strength and the weakness of LANS in more depth, using techniques like meta-learning (Smith-Miles 2009). Third, we are interested in generalizing this approach to other combinatorial optimization problems.

Acknowledgments The authors would like to thank the anonymous reviewers for their insightful comments and suggestions. This work was supported in part by the Fundamental Research Funds for the Central Universities under Grant DUT13RC(3)53, in part by the New Century Excellent Talents in University under Grant NCET-13-0073, in part by China Postdoctoral Science Foundation under Grant 2014M551083, in part by National Program on Key Basic Research Project under Grant 2013CB035906, and in part by the National Natural Science Foundation of China under Grant 61175062 and Grant 61370144.

References

Aiex, R.M., Resende, M.G., Ribeiro, C.C.: TTT plots: a perl program to create time-to-target plots. Optim. Lett. **1**(4), 355–366 (2007)

Alekseeva, E., Kochetov, Y., Plyasunov, A.: Complexity of local search for the p-median problem. Eur. J. Oper. Res. **191**(3), 736–752 (2008)

Avella, P., Sassano, A., Vasil'ev, I.: Computational study of large-scale p-median problems. Math. Program. **109**(1), 89–114 (2007)

Beasley, J.E.: Lagrangean heuristics for location problems. Eur. J. Oper. Res. **65**(3), 383–399 (1993)

Belov, A., Järvisalo, M., Stachniak, Z.: Depth-driven circuit-level stochastic local search for sat. In: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, vol. 1, pp. 504–509. AAAI Press, Menlo Park (2011)

Bennell, J., Song, X.: A beam search implementation for the irregular shape packing problem. J. Heuristics **16**(2), 167–188 (2010)

Brimberg, J., Mladenović, N., Urošević, D.: Local and variable neighborhood search for the k-cardinality subgraph problem. J. Heuristics **14**(5), 501–517 (2008)

Brueggemann, T., Hurink, J.L.: Matching based very large-scale neighborhoods for parallel machine scheduling. J. Heuristics **17**(6), 637–658 (2011)

Ceschia, S., Schaerf, A.: Local search for a multi-drop multi-container loading problem. J. Heuristics **19**(2), 275–294 (2013)

Christofides, N., Beasley, J.E.: A tree search algorithm for the p-median problem. Eur. J. Oper. Res. **10**(2), 196–204 (1982)

Climer, S., Zhang, W.: Searching for backbones and fat: a limit-crossing approach with applications. In: Proceedings of The National Conference on Artificial Intelligence, pp 707–712. AAAI Press, Menlo Park (2002)

Croce, F., Ghirardi, M., Tadei, R.: Recovering beam search: enhancing the beam search approach for combinatorial optimization problems. J. Heuristics **10**(1), 89–104 (2004)

Fischetti, M., Lodi, A.: Local branching. Math. Program. **98**(1–3), 23–47 (2003)

García, S., Labbé, M., Marín, A.: Solving large p-median problems with a radius formulation. INFORMS J. Comput. **23**(4), 546–556 (2011)

Glover, F.: Tabu search-part I. ORSA J. Comput. **1**(3), 190–206 (1989)

Glover, F.: Tabu search-part II. ORSA J. Comput. **2**(1), 4–32 (1990)

Gutin, G., Karapetyan, D.: Local search heuristics for the multidimensional assignment problem. Graph Theory, Computational Intelligence and Thought, pp. 100–115. Springer, Berlin (2009)

Hakimi, S.L.: Optimum locations of switching centers and the absolute centers and medians of a graph. Oper. Res. **12**(3), 450–459 (1964)

Hansen, P., Mladenovic, N.: Variable neighborhood search for the p-median. Locat. Sci. **5**(4), 207–226 (1997)

Helsgaun, K.: General k-opt submoves for the Lin–Kernighan TSP heuristic. Math. Program. Comput. **1**(2–3), 119–163 (2009)

Hoos, H.H., Stützle, T.: Stochastic Local Search. Foundations and Applications. Elsevier, Amsterdam (2005)
Järvinen, P., Rajala, J., Sinervo, H.: A branch-and-bound algorithm for seeking the p-median. Oper. Res. **20**(1), 173–178 (1972)
Kernighan, B., Lin, S.: An eflicient heuristic procedure for partitioning graphs. Bell Syst. Tech. J. **29**, 209 (1970)
Kochetov, Y., Levanova, T., Alekseeva, E., Loresh, M.: Large neighborhood local search for the p-median problem. Yugosl. J. Oper. Res. **15**(1), 53–63 (2005)
Li, C.M., Quan, Z.: An efficient branch-and-bound algorithm based on MaxSAT for the maximum clique problem. In: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, pp. 128–133 (2010)
Mladenovic, N., Brimberg, J., Hansen, P., Moreno-Prez, J.A.: The p-median problem: a survey of metaheuristic approaches. Eur. J. Oper. Res. **179**(3), 927–939 (2007)
Puchinger, J., Raidl, G.R.: Bringing order into the neighborhoods: relaxation guided variable neighborhood search. J. Heuristics **14**(5), 457–472 (2008)
Pullan, W.: A population based hybrid metaheuristic for the p-median problem. In: 2008 IEEE Congress on Evolutionary Computation, pp. 75–82 (2008)
Reese, J.: Solution methods for the p-median problem: an annotated bibliography. Networks **48**(3), 125–142 (2006)
Reinelt, G.: TSPLIB-a traveling salesman problem library. ORSA J. Comput. **3**, 376–384 (1991)
Ren, Z., Jiang, H., Xuan, J., Hu, Y., Luo, Z.: New insights into diversification of hyper-heuristics. IEEE Trans. Cybern (in press) (2013)
Ren, Z., Jiang, H., Xuan, J., Luo, Z.: An accelerated-limit-crossing-based multilevel algorithm for the p-median problem. IEEE Trans. Syst. Man Cybern. B **42**(4), 1187–1202 (2012a)
Ren, Z., Jiang, H., Xuan, J., Luo, Z.: Hyper-heuristics with low level parameter adaptation. Evol. Comput. **20**(2), 189–227 (2012b)
Resende, M.G., Werneck, R.F.: On the implementation of a swap-based local search procedure for the p-median problem. In: Proceedings of the Fifth Workshop on Algorithm Engineering and Experiments, pp. 119–127 (2003)
Resende, M.G.C., Werneck, R.F.: A hybrid heuristic for the p-median problem. J. Heuristics **10**(1), 59–88 (2004)
Riise, A., Burke, E.K.: Local search for the surgery admission planning problem. J. Heuristics **17**(4), 389–414 (2011)
Rosing, K.E., Revelle, C.S., Schilling, D.A.: A gamma heuristic for the p-median problem. Eur. J. Oper. Res. **117**(3), 522–532 (1999)
Senne, E., Lorena, L.: Lagrangean/surrogate heuristics for p-median problems. Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research. Kluwer Academic, Dordrecht (2000)
Smith-Miles, K.A.: Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Comput. Surv. **41**(1), 6:1–6:25 (2009)
Teitz, M.B., Bart, P.: Heuristic methods for estimating the generalized vertex median of a weighted graph. Oper. Res. **16**(5), 955–961 (1968)
Vela, C.R., Varela, R., González, M.A.: Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times. J. Heuristics **16**(2), 139–165 (2010)
Whitaker, R.: A fast algorithm for the greedy interchange for large-scale clustering and median location problems. INFOR J. **21**(2), 95–108 (1983)