A Memetic Genetic Algorithm for the Vertex p-center Problem

Wayne Pullan w.pullan@griffith.edu.au
School of Information and Communication Technology, Griffith University,
Gold Coast, QLD, Australia

Abstract

The p-center problem is one of choosing p facilities from a set of candidates to satisfy the demands of n clients in order to minimize the maximum cost between a client and the facility to which it is assigned. In this article, PBS, a population based meta-heuristic for the p-center problem, is described. PBS is a genetic algorithm based meta-heuristic that uses phenotype crossover and directed mutation operators to generate new starting points for a local search. For larger p-center instances, PBS is able to effectively utilize a number of computer processors. It is shown empirically that PBS has comparable performance to state-of-the-art exact and approximate algorithms for a range of p-center benchmark instances.

Keywords

Memetic genetic algorithms, combinatorial optimization.

1 Introduction

The p-center problem calls for finding the p facilities that minimize the maximum cost of servicing each of n clients, where the pairwise cost of servicing each client from all facilities is given. Each client is only serviced by a single facility and services to clients are not combined. Formally, the p-center problem can be stated as (Mladenović et al., 2003): Let $V = \{v_1, v_2, \ldots, v_n\}$ be a set of n potential locations for facilities and $U = \{u_1, u_2, \ldots, u_m\}$ a set of m users. The distance between (or cost incurred for) each user–facility pair (u_i, v_j) is given as $d_{ij} = d(u_i, v_j)$. The p-center problem is to find $S \subseteq V$ of size p such that $f(S) = \max\{\min d(u_i, v_j)\}, u_i \in U, v_j \in S$ is minimized. The vertex p-center problem addressed in this study is where $V = U$ (that is, all vertices can be either a facility or a user) and is simply referred to as the p-center problem (PC) in the remainder of this article.

PC is a prominent combinatorial optimization problem from location science (Tansel et al., 1983) and has been shown to be NP-hard (Kariv and Hakimi, 1979). Exact techniques for the p-center problem are typically based on the fact that the optimal value for $f(S)$ can be determined by solving a sequence of different, auxillary, subproblems. Minieka (1970) proposed an algorithm that involved solving a finite series of set covering problems to identify the minimum threshold distance at which all customers are covered using p facilities. Based on this algorithm, Daskin (1995) developed an algorithm that found the minimum threshold distance using a bisection method which systematically reduced the gap between the upper and lower bounds of the optimal solution. This algorithm was further developed in Daskin (2000) where a maximum set covering subproblem was formulated and solved using Langrangian relaxation. Ilhan and Pınar (2001) proposed a method that first finds a lower bound to the optimal solution for the problem by solving a series of feasibility problems based on a linear programming formulation and then, using an integer programming technique, uses feasibility problems to check for feasible solutions. The exact algorithm of Elloumi et al. (2004) solved the set covering subproblems using a greedy heuristic and an integer programming formulation of the subproblem. More recently, Al-Khedhairi and Salhi (2005) developed modifications to both the Daskin (1995) and Ilhan and Pınar (2001) algorithms that improved their efficiency.

Unlike the other basic discrete location problem, the p-median problem (Resende and Werneck, 2004; Cranic et al., 2004; Taillard, 2003; Hansen et al., 2001), few heuristic approaches to solving the p-center problem have been proposed. Initially, heuristics (Hochbaum and Shmoys, 1985; Martinich, 1988; Plesnik, 1987) exploited the close relationship between the p-center problem and the dominating set problem. After a gap of 10 years the next, and most recent, meta-heuristic for the p-center problem was proposed by Mladenović et al. (2003) where local search was combined with the meta-heuristics: *Multi-Start*, *Tabu Search*, and *Variable Neighbourhood Search* (VNS) and compared with three classical heuristics *Greedy*, *Alternate*, and *Interchange* (the most common heuristics used for the p-median problem) in addition to the *Binary Search* heuristic (Hochbaum and Shmoys, 1985). The conclusion reached was that, overall, the combination of VNS with local search had the best performance of the heuristics examined. In Hassin et al. (2003), a local search strategy is proposed that is directed at combinatorial optimization problems with a min-max (or max-min) objective. Using this approach, solutions are compared lexicographically rather than by their worst coordinate and they applied this local search strategy to a number of small p-center problems. Previous theoretical studies of problems related to the p-center problem include Krumke (1995) and Cheng et al. (2007) while Scaparra et al. (2004) investigates the capacitated p-center problem.

This article describes a memetic genetic algorithm, dubbed PBS, for the p-center problem, which incorporates phenotype crossover and directed mutation operators, to generate new starting points for a local search. For larger p-center instances, PBS is able to effectively utilize a number of computer processors. Based on computational experiments, this study demonstrates that PBS has excellent performance with regard to robustness, quality of results, and computational requirements, on a range of PC benchmark instances, and hence represents an improvement in heuristic PC solving algorithms. Descriptive results on the behavior of the components of PBS are also presented that give insights into the roles of these components during the search.

The remainder of this article is structured as follows. The PBS algorithm is first described. Next, empirical performance results are presented that establish PBS as an effective meta-heuristic for solving the PC problem. This is followed by a more detailed investigation of the behavior of PBS and the factors determining its performance. Finally, the main contributions of this work are summarized, insights gained from this study are presented, and some directions are outlined for future research.

2 The PBS Algorithm

The PBS algorithm is based on the fundamental concept of augmenting a population based meta-heuristic with an iterative improvement local search. The goal of the meta-heuristic is to generate a wide range of "good" starting points for the local search while the role of the local search is to effectively explore regions that are relatively close to these starting points. Key features of the PBS algorithm are now described using the following terminology and syntax: $G = (V, E)$—a directed or undirected graph, where V is the set of all n vertices and E is the set of all possible edges; S—p-center solution ($S \subseteq V$); p—required number of facilities in S; facility—$v \in S$; S_c—$\max\{\min d(v_i, v_j)\}$, $v_i \in V$, $v_j \in S$, that is, the maximum cost of servicing vertices in V from the facilities in S. Each vertex is serviced by a single facility and the cost of servicing all vertices from all facilities is predefined for each instance; $|S|$—the number of facilities currently in S; P—PBS population of p-center solutions; P_i—a p-center solution within the population P; N_v—distance ordered list of vertices that are neighbors of vertex v (note that $N_v[0] = v$); N_{vk}—a sublist containing the first k elements of N_v ($= \{N_v[0], \ldots, N_v[k-1]\}$); and, N_p—the number of computer processors to be used by PBS.

As an overview and with reference to the high-level pseudo-code description of the PBS algorithm that follows, PBS uses a relatively small population P of eight p-center solutions (initially, randomly generated ($\mathcal{C}$) and optimized using the local search ($\mathcal{L}$)) to provide inputs to genetic mutation ($\mathcal{M}_1, \mathcal{M}_2$) and crossover ($\mathcal{X}_1, \mathcal{X}_2$) operators. The outputs from these operators subsequently become inputs to the local search which produces candidate solutions for addition to P. This process is repeated until either a target cost is attained or the maximum allowed number of generations is achieved.

Algorithm PBS

```
        (Create population $P$)
1.      $\forall P_i \in P, P_i \leftarrow \mathcal{L}(\mathcal{C})$
2.      **while not finished**
            (Perform each generation)
3.          $\forall P_i \in P$
4.              $\forall P_j \in P - P_i$
5.                  $P \leftarrow P + \mathcal{L}(\mathcal{M}_1(P_j))$
6.                  $P \leftarrow P + \mathcal{L}(\mathcal{M}_2(\mathcal{X}_1(P_i, P_j)))$;
7.                  $(S_1, S_2) \leftarrow \mathcal{X}_2(P_i, P_j)$;
8.                  $P \leftarrow P + \mathcal{L}(\mathcal{M}_2(S_1)) + \mathcal{L}(\mathcal{M}_2(S_2))$;
```

In more detail, at the commencement of PBS (Line 1), all the initial solutions for P are created ($\mathcal{C}$) by setting each P_i to a single vertex chosen uniformly at random from the given graph G. Then the local search ($\mathcal{L}$) is invoked to first increase the number of facilities in P_i to p after which an iterative improvement phase is entered and facility-vertex swaps are performed for some number of iterations. Once the initial population has been created, PBS repeatedly performs generations (Lines 2–8) during which crossovers ($\mathcal{X}_1, \mathcal{X}_2$) between all possible combinations of (P_i, P_j) and $|P|(|P|-1)$ $\mathcal{M}_1$ mutations of each $P_i \in P$ are performed. In addition, $\mathcal{M}_2$ mutations are performed on all child solutions generated by the $\mathcal{X}_2$ crossover. Immediate updating of P is performed whenever a lower cost p-center solution is found. At the completion of each crossover and mutation operation, child solutions S are locally optimized using the local search $\mathcal{L}$ which, if $|S| = p$ iteratively removes a facility and replaces it with the vertex, the result is the maximum decrease/smallest increase in the cost of S. If $|S| < p$, then the local search simply adds the vertex, which results in the largest decrease in the cost of S. To maintain diversity in the population, no new p-center solution S is added ($+$) to P if it is close (in cost or facilities) to any P_i in P.

PBS is able to execute in either a single processor mode, as described in the PBS algorithm above (referred to as PBS-1 in this article), or in a parallel processing mode (referred to as PBS-x in this article, where x is the number of computer processors utilized). PBS-x implements the PBS algorithm using a simple master-slave model of parallel processing where the single master task controls the distribution of pairs of solutions from P to each slave task and also the potential updating of the population P when the slave tasks return the results from all crossover and mutation operators (and the subsequent local searches). That is, for PBS-x, all $\mathcal{C}$, $\mathcal{X}_1$, $\mathcal{X}_2$, $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{L}$ operations are performed in parallel in the slave tasks, thus providing an x-way overlapping of the mutation, cross-over, and local search portions of the PBS algorithm. Communication between the master and each slave task and control of the slave tasks by the master task is performed using messages passed through the MPI message passing interface (`http://www-unix.mcs.anl.gov/mpi/`). A copy of the code for PBS can be obtained by contacting the author.

2.1 Mutation Operators $\mathcal{M}_1$ and $\mathcal{M}_2$

Within PBS, two mutation operators exist. The first, $\mathcal{M}_1$, is a random mutation operator that takes a p-center solution P_i and constructs a child solution S by first randomly selecting some number q in the range $p/2, \ldots, p$ and then using q randomly selected facilities from P_i and $p - q$ randomly selecting vertices from $V - P_i$ to construct S. The local search is then invoked using S as the starting point and the resulting p-center solution becomes a candidate for addition to P. The second mutation operator ($\mathcal{M}_2$) is a directed mutation operator that, based on the assumption that facilities should not be "close" to each other, identifies the closest pair of facilities within P_i, deletes them from P_i, and then invokes local search to generate a new p-center solution which becomes a candidate to be added to P.

2.2 Crossover Operators $\mathcal{X}_1$ and $\mathcal{X}_2$

The specific crossover operators implemented within PBS are: $\mathcal{X}_1$, a random crossover operator which constructs a single child solution by randomly selecting p facilities from the two parents; and $\mathcal{X}_2$, which first randomly selects two users and a random number $q \in [0.1, \ldots, 0.9]$. Let d_{i1} be the distance between facility i and the first selected user and d_{i2} the distance between facility i and the second selected user. The first child solution is constructed by combining all facilities i in the first parent where $d_{i1}/d_{i2} \leq q$ with all facilities j in the second parent where $d_{j1}/d_{j2} > q$. The second child solution is built by combining all facilities i in the first parent where $d_{i1}/d_{i2} > q$ with all facilities j in the second parent where $d_{j1}/d_{j2} \leq q$. For this crossover operator, if more than p facilities are added to a child solution then the required number are randomly removed to reduce the total down to p. If less than p facilities are present then the subsequent local search will add the required number to bring the child solution back to p facilities. The motivation for this crossover operator is that, one of the primary functions of crossover genetic operators is the identification of "good" building blocks within parent solutions and the extraction of these into the children solutions. Clearly these good building blocks will tend to exist in the p-center problem as groups of facilities that are, physically, relatively close to each other and this physical relationship is not preserved within the p-center solution S. The $\mathcal{X}_2$ operator, which operates in the physical space, does create building blocks of physically close facilities whereas the $\mathcal{X}_1$ operator, which operates directly on the S vector, does not. It should be noted that equivalent phenotype crossover operators have been very successfully applied to clustering problems in other domains where there is a distance dependent interaction between the elements of the problem [e.g., the structural optimization of atomic clusters (Pullan, 2005)].

2.3 Local Search $\mathcal{L}$

The local search component of PBS takes a p-center solution S and, if $|S| < p$, adds facilities to S by uniform randomly selecting a facility from N_{wk} where w is a vertex whose distance from the nearest facility is currently the maximum (i.e., S_c) and k is the index in N_w of the facility that currently services vertex w. Once $|S| = p$, the local search enters a swap phase where, for each iteration of the local search, a facility is removed from S and replaced by a vertex that has not previously been swapped with that facility. Selection of the vertex to be added to S is performed by evaluating, for each vertex in N_{wk}, the overall changes in S_c by swapping this vertex with each facility in S. The actual vertex and facility to be swapped is that pair which provides the best decrease (or least increase) in p-center cost. To prevent the local search cycling around vertices, once a facility/vertex swap has been performed, that combination is no longer eligible for selection during the current invocation of the local search. The termination criteria for the local search is either $2n$ iterations or no decrease in S_c for $0.1(g + 1)n$ iterations (where g is the PBS generation count). Experiments, not reported in this article, were performed to determine the form of and values used in these termination criteria.

The local search was primarily designed to keep the time-complexity of the individual search steps minimal which is in keeping with the common intuition that, in the context of stochastic local search algorithms, it is often preferable to perform many relatively simple, but efficiently computable search steps rather than fewer complex search steps. As the local search function of PBS is the major computational component of PBS, an efficient algorithm is essential. The PBS data structures that allow the efficient implementation of local search are based on those used in the VNS meta-heuristic (Mladenović et al., 2003) (which in turn are based on those developed by Whitaker, 1983), and are as follows: F_v^0—facility which is closest to vertex v; D_v^0—distance from vertex v to the closest facility; F_v^1—facility which is second closest to vertex v; D_v^1—distance from vertex v to the second closest facility. Using these data structures and the previously defined notation, the three key functions within local search are now described:

Procedure Add_Facility(f)

```
1.  $S_c \leftarrow 0$
2.  $S \leftarrow S \cup f$
3.  $\forall\, v \in V$
4.      if $d_{fv} < D_v^0$
5.          $D_v^1 \leftarrow D_v^0$; $F_v^1 \leftarrow F_v^0$; $D_v^0 \leftarrow d_{fv}$; $F_v^0 \leftarrow f$
6.      else if $d_{fv} < D_v^1$
7.          $D_v^1 \leftarrow d_{fv}$; $F_v^1 \leftarrow f$
8.      if $D_v^0 > S_c$
9.          $S_c \leftarrow D_v^0$
```

Procedure **Add_Facility** has worst-case complexity $\mathcal{O}(n)$ and adds facility f to the p-center solution S, updates the data structures and determines the cost S_c of S.

Procedure Remove_Facility(f)

```
1.  $S_c \leftarrow 0$
2.  $S \leftarrow S - f$
3.  $\forall\, v \in V$
4.      if $F_v^0 = f$
```

```
5.            $D_v^0 \leftarrow D_v^1$; $F_v^0 \leftarrow F_v^1$; $(D_v^1, F_v^1) \leftarrow$ **Find_Next**($v$)
6.        **else if** $F_v^1 = f$
7.            $(D_v^1, F_v^1) \leftarrow$ **Find_Next**($v$)
8.        **if** $D_v^0 > S_c$
9.            $S_c \leftarrow D_v^0$
```

Procedure **Remove_Facility** removes facility f from the p-center solution S, updates the data structures, and determines the cost S_c of S. The function **Find_Next(v)** scans S to find the second closest facility to v after f has been removed from S. While **Remove_Facility** has worst-case complexity $\mathcal{O}(np)$, from a practical point of view, it is extremely unlikely that this will occur as it would require facility f to be either the closest or second closest facility to all $v \in V$. The other mitigating factor is that, as described below, **Remove_Facility** is used only once per iteration of the local search.

```
Function Find_Pair($w$)
 1.   $C \leftarrow \max(d_{ij})$
 2.   $\forall\, i \in N_{wk}$
 3.       **Add_Facility($i$)**
 4.       $\forall\, f \in S, M_f \leftarrow 0$
 5.       $\forall\, v \in V - S$
 6.           **if** $\min(d_{iv}, D_v^1) > M_{F_v^0}$
 7.               $M_{F_v^0} \leftarrow \min(d_{iv}, D_v^1)$
 8.       $\forall\, f \in S$
 9.           **if** $M_f = C$
10.               $L \leftarrow L \cup (f, i)$
11.           **else if** $M_f < C$
12.               $L \leftarrow (f, i)$; $C \leftarrow M_f$
13.       **Remove_Facility($i$)**
14.   **return** $(f, i) \in L$
```

Function **Find_Pair** finds an optimal (facility, vertex) pair to swap. Clearly, to improve S_c, a vertex in N_{wk}, where $D_w^0 = S_c$ and k is the index of F_w^0 in N_w, must be added to S. Function **Find_Pair** evaluates, for all vertices in N_{wk}, the changes in S_c caused by adding that vertex and removing, in turn, all existing facilities in S (in the algorithm description for **Find_Pair**, M_f represents the cost incurred when facility f is removed from S). A list L of best decrease/smallest increase (facility, vertex) pairs is maintained and the actual pair to be swapped is uniform randomly selected from this list. Function **Find_Pair** has the potential of being $\mathcal{O}(n^2)$ however, in practice, this is mitigated as, while k may be large for the initial randomly generated solutions, it rapidly becomes relatively small. In addition, for large problems, a fixed limiting value can be placed on k so that function **Find_Pair** becomes $\mathcal{O}(n)$. A further optimization that was made is to replace the invocation of procedure **Remove_Facility** with a simple memory copy of relevant data structures saved prior to the invocation of function **Find_Pair**.

3 Empirical Performance Results

In order to evaluate the performance and behavior of PBS, computational experiments were performed on a benchmark of PC instances (in this context, a PC instance is defined as the combination of a graph G and a particular p value). This benchmark consisted of

40 instances from the *ORLIB*[1] class (Beasley, 1985) and 98 instances from the *TSPLIB*[2] class (Reinelt, 1991) giving a total of 138 instances having between 100 and 3038 vertices with p in the range $5, \ldots, 500$.

All experiments for this study were performed on AMD Opteron 252 2.6GHz computer processors that, when executing the DIMACS Machine Benchmark (*dmclique*[3]) on a single processor, required 0.31 CPU seconds for r300.5, 1.93 CPU seconds for r400.5, and 7.35 CPU seconds for r500.5. Note that, as PBS is able to execute using a number of computer processors, in the following sections, PBS-x denotes an x computer processor instance of PBS.

In the following sections, results from a series of experiments that were aimed at providing a detailed assessment of the performance of PBS are first presented. Then, additional experimental results that facilitate a more direct comparison between PBS and VNS (Mladenović et al., 2003) are reported.

3.1 PBS Performance

To evaluate the performance of PBS on the instances identified above, extended trials were first performed on those instances where an optimal p-center cost was unknown to determine if PBS could improve on the "best known cost" reported to date in the literature. Then, some number (10 or 100) independent trials of PBS were performed for each instance, using a target p-center cost identical to the respective optimal or best known cost. The results from these experiments are displayed in Tables 1 and 2 along with the results obtained in previous studies for the exact algorithms ELP (Elloumi et al., 2004), IP* (Al-Khedhairi and Salhi, 2005), Daskin* (Al-Khedhairi and Salhi, 2005), and the meta-heuristic VNS (Mladenović et al., 2003). As PBS uses floating point data and the exact algorithms that it is compared with use integer data, a result obtained by PBS is deemed to be identical to the result obtained by an exact algorithm if the rounded PBS result equals the exact algorithms result. It should be noted that, for PBS, the CPU times do not include the initial reading of input data files nor the creation of static data structures. In addition, for any trial in which PBS did not attain the target p-center cost, the reported CPU/elapsed time recorded is that at which the minimum p-center cost was attained.

In summary, PBS-1 finds optimal solutions with a success rate of 100% over 100 trials (that is, all trials were successful) for all instances of the *ORLIB* class (Table 1) and the small *TSPLIB* instances (*pr226*, *pr264*, *pr299*, *pr439*, *pcb442*, *kroA200*, *kroB200*, *lin318*, *gr202*, *d493*, and *d657*). For the *u1060* and *u1817* classes (Table 2), with 15 instances each, PBS-1 found all optimal solutions with a success rate of 100% over 10 trials, produced five new, best known results for *u1817*, but failed to obtain one currently best known *u1817* result. For the 10 instances of the *rl1323* class, the PBS-1 success rate was 100% over 10 trials for six instances, 80% with an average percentage deviation of 0.11% for three instances, and failed on the remaining *rl1323* instance with an average percentage deviation of 0.34%. For the *pcb3038* class (Table 2), with 14 instances, PBS-16 found 14 new best known costs. These results, over a range of instances, vertices, and facilities, support the claim that PBS is robust and also capable of obtaining high quality results.

[1]`http://people.brunel.ac.uk/~mastjjb/jeb/info.html`

[2]`http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`

[3]`ftp://dimacs.rutgers.eduindirectory/pub/dsj/clique`

Table 1: Unscaled CPU Times for ELP (Elloumi et al., 2004), IP* (Al-Khedhairi and Salhi, 2005), Daskin* (Al-Khedhairi and Salhi, 2005), Unscaled Average CPU Times for VNS (Mladenović et al., 2003), and PBS-1 for the *ORLIB* and Small *TSPLIB* Classes. PBS-1 Was 100% Successful for All 100 Trials for Each Instance of *ORLIB* and Small *TSPLIB* Classes. STD Is the Standard Deviation of the CPU Times for PBS-1 over the 100 Trials for Each Instance. CPU and STD Values < 0.01 Are Shown as $< \epsilon$,—Signifies that No Results Are Available for the Specified Opt. Value, and the PBS-1 Best Column Shows the Target Cost Used for PBS-1

ORLIB										Small *TSPLIB*								
				CPU				PBS-1						PBS-1	CPU		PBS-1	
File	n	p	Opt	ELP	IP*	Daskin*	VNS	CPU	STD	Class	n	p	Opt	Best	IP*	Daskin*	CPU	STD
pmed01	100	5	127	0.7	4.05	2.09	0.04	$< \epsilon$	$< \epsilon$	pr226	226	40	650	65$< \epsilon$	7.39	5.68	$< \epsilon$	$< \epsilon$
pmed02	100	10	98	0.2	1.52	1.45	4.45	0.01	0.01	pr226	226	20	1366	1365.65	9.55	7.28	$< \epsilon$	$< \epsilon$
pmed03	100	10	93	0.1	1.81	1.35	0.17	0.06	0.05	pr226	226	10	2326	2326.48	8.85	9.29	0.03	0.03
pmed04	100	20	74	0.1	1.01	0.92	0.30	$< \epsilon$	0.01	pr226	226	5	3721	3720.55	19.92	8.08	0.01	0.02
pmed05	100	33	48	0.1	1.49	0.73	0.11	$< \epsilon$	$< \epsilon$	pr264	264	40	316	316.23	—	—	$< \epsilon$	0.01
pmed06	200	5	84	0.3	13.53	9.01	1.51	0.02	0.02	pr264	264	20	515	514.78	11.19	9.50	0.02	0.02
pmed07	200	10	64	0.5	5.09	4.31	1.30	0.01	0.02	pr264	264	10	850	85$< \epsilon$	12.00	11.73	0.01	0.01
pmed08	200	20	55	0.4	5.31	3.34	2.22	0.01	0.01	pr264	264	5	1610	1610.12	14.62	16.27	$< \epsilon$	$< \epsilon$
pmed09	200	40	37	0.1	3.46	2.66	10.89	$< \epsilon$	$< \epsilon$	pr299	299	40	355	355.52	12.95	11.70	0.18	0.16
pmed10	200	67	20	0.3	2.76	2.57	9.04	$< \epsilon$	$< \epsilon$	pr299	299	20	559	559.02	14.29	13.10	0.22	0.23
pmed11	300	5	59	1.0	11.67	16.25	1.52	0.04	0.06	pr299	299	10	889	888.84	23.95	18.62	0.43	0.36
pmed12	300	10	51	1.3	12.03	12.25	5.39	0.01	0.02	pr299	299	5	1336	1336.27	18.24	18.74	0.09	0.11
pmed13	300	30	36	0.8	14.43	8.23	10.21	0.05	0.05	pr439	439	40	672	671.75	32.52	33.03	0.66	0.76
pmed14	300	60	26	0.9	6.61	6.81	70.67	0.01	0.01	pr439	439	20	1186	1185.59	35.11	29.21	0.12	0.14
pmed15	300	100	18	1.0	4.43	4.40	55.86	$< \epsilon$	$< \epsilon$	pr439	439	10	1972	1971.83	39.96	36.16	0.09	0.12
pmed16	400	5	47	1.6	30.01	28.10	0.08	0.01	$< \epsilon$	pr439	439	5	3197	3196.58	53.08	51.74	1.84	2.02
pmed17	400	10	39	2.1	30.88	27.06	26.00	0.02	0.02	pcb442	442	40	316	316.23	2714.00	1302.00	0.04	0.04
pmed18	400	28	28	1.4	12.49	13.17	—	0.13	0.12	pcb442	442	20	447	447.21	37.62	151.20	0.89	0.95
pmed19	400	80	18	0.4	9.89	10.16	—	1.08	1.01	pcb442	442	10	671	670.82	34.30	128.80	0.27	0.33
pmed20	400	133	13	1.8	13.53	9.30	—	0.10	0.08	pcb442	442	5	1025	1024.74	40.31	41.37	0.47	0.52
pmed21	500	5	40	5.2	56.40	56.01	1.24	0.01	0.01	kroA200	200	40	258	258.26	5.19	5.02	0.14	0.12

Table 1: Continued

ORLIB									
				CPU				PBS-1	
File	n	p	Opt	ELP	IP*	Daskin*	VNS	CPU	STD
pmed22	500	10	38	4.3	495.20	60.78	82.63	1.12	1.18
pmed23	500	50	22	1.2	28.52	16.45	—	2.11	1.84
pmed24	500	100	15	4.5	14.64	12.59	—	0.06	0.04
pmed25	500	167	11	2.7	13.06	10.28	—	0.05	0.04
pmed26	600	5	38	6.1	401.60	104.20	0.58	0.03	0.03
pmed27	600	10	32	8.2	78.24	65.23	5.07	0.04	0.04
pmed28	600	60	18	2.1	39.51	19.31	—	0.13	0.11
pmed29	600	120	13	5.1	32.00	23.61	762.44	0.05	0.03
pmed30	600	200	9	5.4	34.72	17.22	—	0.80	0.60
pmed31	700	5	30	8.1	303.00	122.60	0.58	0.03	0.01
pmed32	700	10	29	45.2	1447.00	116.80	165.26	0.31	0.44
pmed33	700	70	15	3.1	94.07	33.11	—	81.75	107.25
pmed34	700	140	11	6.5	50.23	29.66	—	0.04	0.01
pmed35	800	5	30	13.7	183.90	123.30	6.67	0.10	0.12
pmed36	800	10	27	34.5	3602.00	110.50	105.99	0.96	0.97
pmed37	800	80	15	2.0	105.80	49.02	—	0.27	0.21
pmed38	900	5	29	18.5	251.00	273.10	1.92	0.03	0.01
pmed39	900	10	23	27.3	5817.00	208.70	—	26.30	27.12
pmed40	900	90	13	7.8	240.80	462.90	—	0.46	0.29
Total				226.60	13475.00	2079.00		116.21	

Small *TSPLIB*								
				PBS-1	CPU		PBS-1	
Class	n	p	Opt	Best	IP*	Daskin*	CPU	STD
kroA200	200	20	389	389.31	5.51	5.38	0.12	0.11
kroA200	200	10	599	598.82	6.57	6.83	0.58	0.61
kroA200	200	5	911	911.41	8.22	8.28	0.10	0.12
kroB200	200	40	253	253.24	5.08	4.29	0.04	0.04
kroB200	200	20	382	382.28	8.27	4.75	0.04	0.03
kroB200	200	10	582	582.10	6.69	5.84	0.05	0.05
kroB200	200	5	898	897.67	7.63	7.53	0.01	0.01
lin318	318	40	316	315.92	219.40	11.67	0.01	0.01
lin318	318	20	496	496.45	13.54	14.49	15.92	20.18
lin318	318	10	743	743.21	17.46	18.52	0.47	0.50
lin318	318	5	1101	1101.34	18.68	22.06	0.26	0.35
gr202	202	40	3	2.97	5.15	3.81	0.05	0.05
gr202	202	20	6	5.97	5.65	4.63	0.01	0.01
gr202	202	10	9	9.33	6.79	4.89	0.05	0.07
gr202	202	5	19	19.38	8.59	5.83	$< \epsilon$	$< \epsilon$
d493	493	40	206	206.02	45.38	79.97	36.25	32.78
d493	493	20	313	312.74	1406.00	75.54	19.36	21.75
d493	493	10	458	458.30	60.13	82.25	5.21	4.79
d493	493	5	753	752.91	72.24	75.41	19.96	15.86
d657	657	40	250	249.52	3126.00	351.70	196.90	354.34
d657	657	20	375	374.70	301.15	255.80	5.30	5.07
d657	657	10	575	574.74	751.01	156.60	28.58	25.70
d657	657	5	881	880.91	100.56	154.70	209.97	28.66
Total					9352.00	3269.00	544.75	

Table 2: Unscaled CPU Times for ELP (Elloumi et al., 2004), Average CPU Times for PBS-1 and Average Elapsed Times for PBS-16 for the Large *TSPLIB* Classes. PBS-1 and PBS-16 Results Are from 10 Consecutive Trials. The ELP Opt. Column Shows the Optimal Cost Obtained by ELP; However, Where an Optimal Cost Was Not Identified by ELP, the "?" Signifies the Best Found Solution. The PBS-1 Best and PBS-16 Best Columns Show the Target Costs Used for PBS-1 and PBS-16, Respectively. PBS-1 Best Figures in **Bold** Signify New Best Known Results. VNS Best Shows the Best Results Obtained by VNS (Mladenović et al., 2003). STD Is the Standard Deviation of the CPU Times Over the 10 Trials for Each Instance. For All Classes, the %Dev. Column Shows the Average PBS Percentage Deviation from the PBS Best Cost When Not All 10 Trials Were Successful in Obtaining the PBS Best Cost for an Instance

			ELP	PBS-1	ELP	PBS-1						ELP	PBS-1	ELP	PBS-1		
Class	n	p	Opt.	Best	CPU	CPU	SD	%Dev	Class	n	p	Opt.	Best	CPU	CPU	SD	%Dev.
u1060	1060	10	2273	2273.08	53	138.11	56.30	0.00	rl1323	1323	10	3077	3077.30	1,380	4,760.10	1905.62	0.00
u1060	1060	20	1581	1580.80	2778	659.44	1449.55	0.00	rl1323	1323	20	2016	2016.40	480	605.90	226.38	0.00
u1060	1060	30	1208	1207.77	298	36.84	25.89	0.00	rl1323	1323	30	1632	1631.50	900	1,200.20	1138.23	0.00
u1060	1060	40	1021	1020.56	366	47.73	43.65	0.00	rl1323	1323	40	1352	1352.36	3,000	292.00	154.02	0.00
u1060	1060	50	905	904.92	383	233.13	129.32	0.00	rl1323	1323	50	1187	1187.27	8,580	619.40	211.66	0.00
u1060	1060	60	781	781.17	233	103.12	74.46	0.00	rl1323	1323	60	1063	1063.01	9,120	8,184.90	8206.12	0.03
u1060	1060	70	711	710.76	135	109.56	20.23	0.00	rl1323	1323	70	972	971.93	1,740	7,427.00	4157.30	0.03
u1060	1060	80	652	652.16	60	142.11	54.56	0.00	rl1323	1323	80	895	895.06	420	8,783.00	6698.80	0.28
u1060	1060	90	608	607.88	38	63.15	27.14	0.00	rl1323	1323	90	832	832.00	120	929.90	584.37	0.00
u1060	1060	100	570	570.01	29	17.54	16.43	0.00	rl1323	1323	100	787	789.70	120	1,840.50	1070.96	0.34
u1060	1060	110	539	538.84	30	160.73	74.35	0.00	**Total**					25,860	34,642.90		
u1060	1060	120	510	510.28	44	107.65	28.90	0.00									
u1060	1060	130	500	499.65	44	118.71	77.53	0.00									
u1060	1060	140	452	452.46	46	318.48	150.04	0.00									
u1060	1060	150	447	447.01	50	10.59	12.87	0.00									
Total					4587	2266.89	0.00										

Table 2: Continued

			ELP	PBS-1	ELP	PBS-1						VNS	PBS-16			
Class	n	p	Opt.	Best	CPU	CPU	STD	%Dev	Class	n	p	Best	Best	Elapsed	STD	%Dev.
u1817	1817	10	458	457.91	2,700	5,316.60	728.10	0.00	pcb3038	3038	10	—	728.54	5415.40	1052.22	2.15
u1817	1817	20	310?	**309.01**	4,920	10,243.00	3,842.53	0.00	pcb3038	3038	20	—	493.04	972.10	386.60	1.46
u1817	1817	30	250?	**240.99**	16,500	1,605.50	371.02	0.00	pcb3038	3038	30	—	393.50	438.60	57.12	1.80
u1817	1817	40	210?	**209.46**	6,420	193.70	206.73	0.00	pcb3038	3038	40	—	336.42	763.00	225.98	0.51
u1817	1817	50	187?	**184.91**	9,840	1,128.90	714.66	0.00	pcb3038	3038	50	307.48	298.20	731.60	330.59	0.07
u1817	1817	60	163	162.65	1,260	837.30	287.25	0.00	pcb3038	3038	100	215.67	206.63	565.80	233.09	0.87
u1817	1817	70	148	148.11	420	191.80	193.45	0.00	pcb3038	3038	150	174.83	164.77	427.00	188.18	1.82
u1817	1817	80	137	136.80	1,140	127.50	105.50	0.00	pcb3038	3038	200	157.00	140.90	899.40	277.27	1.34
u1817	1817	90	130?	129.54	7,202	2,963.50	2646.63	0.00	pcb3038	3038	250	137.54	122.78	723.70	114.91	1.53
u1817	1817	100	127	127.01	300	146.40	142.66	0.00	pcb3038	3038	300	123.33	115.25	537.40	110.29	1.66
u1817	1817	110	109	109.25	420	13,772.40	16,610.21	0.00	pcb3038	3038	350	118.02	104.81	805.90	153.06	1.88
u1817	1817	120	108	107.78	120	80.10	62.16	0.00	pcb3038	3038	400	107.65	97.51	620.60	119.25	2.01
u1817	1817	130	108?	107.75	3,720	11.20	6.46	0.00	pcb3038	3038	450	101.51	88.96	666.20	122.86	2.19
u1817	1817	140	105?	**101.61**	4,020	4,949.30	2,349.44	0.00	pcb3038	3038	500	94.37	85.00	435.80	87.38	1.18
u1817	1817	150	94?	101.60	5,640	314.00	54.35	0.00								
Total					64,622	41,881.20										

3.2 Comparative Results

The results reported in the previous section demonstrate that PBS achieves excellent performance on standard benchmark instances. However, a comparative analysis of these results, as compared to the results found in the literature for other state-of-the-art heuristic PC algorithms, is not a straightforward task as comparisons must take into account the two related metrics of the computational requirements of the algorithms and also the quality of the results produced.

The computational requirements of PBS have been documented in terms of the CPU time/elapsed time used on a computer processor that has been benchmarked using a publicly available benchmark program. However, in previous studies, computing hardware has basically been documented in terms of CPU type (and, sometimes, CPU clock rate) which only allows a very basic means of comparison (i.e., by scaling based on the computer CPU clock rate which, e.g., takes no account of other features, such as memory caching, memory size, hardware architecture, etc.). In addition, differences in what is actually included in the measured times need to be accounted for. With regard to the previous studies referenced in this article, the computer processor used to obtain the ELP (Elloumi et al., 2004) results shown in Tables 1 and 2 was a 400 MHz Pentium II, for IP* and Daskin* (Al-Khedhairi and Salhi, 2005) a Sun Enterprise Workstation 450 was utilized while for VNS (Mladenović et al., 2003) a Sun Sparc Station 10 was employed.

With regard to the quality of results metric, it must be noted that seemingly small differences in the average p-center costs and average percentage cost deviation may in fact represent major differences in algorithm performance, since (as in many hard combinatorial optimization problems) finding p-center solutions with slightly higher costs is typically substantially easier than finding those with minimal costs. For example, for the *u1060*, $p = 130$ instance, achieving an average cost of 500.14 required an average of 5.60 CPU seconds (Table 5) for PBS-1 while achieving an average cost of 499.65 required, on average, 118.71 CPU seconds (Table 2). Basically, the two metrics of computational requirements and the quality of results are related by a nonlinear relationship which can cause the computational requirements to increase exponentially as the quality of the result improves.

3.2.1 Comparison with Exact Algorithms

Given the lack of definition of computer processors, an objective comparison of the CPU requirements of PBS with those of the exact algorithms (ELP, IP*, and Daskin*) is basically not possible, however it can be noted that the ranking of the instances, in terms of CPU requirements, is different for each algorithm. With regard to the quality of results, Table 3 provides a summary, for each of the three exact algorithms and PBS, as to the number of optimal or best known solutions found for the 138 instances of the benchmark used in this study. There is a total of 116 instances for which the optimal result is known while best known results only are available for the remaining 22 instances. PBS obtained 112 of the 116 optimal results with a success rate of 100%, three of the remaining four with a success rate of 80%, and failed completely to locate the one remaining optimal result (*rl1323* with $p = 100$). Of the 22 best known results, PBS alone located 19, both PBS and ELP located two more, and only ELP located the one remaining best known result.

3.2.2 Comparison with VNS

To enable a more direct comparison with VNS (Mladenović et al., 2003), additional PBS-1 trials were performed in which the target p-center cost was set to those obtained

Table 3: Summary Counts for the Number of Optimal or Best Known Results Found for the Exact Algorithms and PBS for the 138 Instance Benchmark Used in This Study. Opt. Is Optimal Result Known; BK Is Best Known Result Only; — Signifies That an Algorithm Was Not Applied to That Class

	Solution		Solutions found			
Class	Type	Count	ELP	IP*	Daskin*	PBS
ORLIB	Opt.	40	40	40	40	40
Small *TSPLIB*	Opt.	44	—	44	44	44
u1060	Opt.	15	15	—	—	15
rl1323	Opt.	10	10	—	—	9
u1817	Opt.	7	7	—	—	7
u1817	BK	8	3	—	—	7
pcb3038	BK	14	—	—	—	14

for VNS. While this technique is inherently biased toward VNS (as all PBS-1 results will be less than or equal to the average obtained for VNS and hence the PBS-1 average will be less than that for VNS) it does reduce the impact of the differences in the quality of results on the comparisons and the CPU related differences become the major issue when comparing results obtained by PBS with those obtained for VNS.

Table 4 shows the results obtained where, for all the *ORLIB* instances, PBS-1 was run over 100 trials with a target cost that produced *p*-center costs identical to those reported for VNS. The average overall ratio of the average CPU times required by PBS-1 to those reported for VNS is 0.0007. Table 5 shows the results obtained where, for the *TSPLIB* class *u1060*, PBS-1 was run over 100 trials with a target cost set equal to the average VNS cost (Mladenović et al., 2003) for this class. The average overall ratio of the average CPU times required by PBS-1 to those reported for VNS (Mladenović et al., 2003) is 0.0270 for the *u1060* instances and 0.1107 for the *pcb3038* instances. The average of these three ratios is 0.0461; however, it is not possible to determine exactly what proportion of this improvement can be attributed to algorithmic differences rather than differences in computing hardware.

3.2.3 Conclusion

While these comparisons with other studies suggest that PBS has excellent CPU performance, no final conclusive statement can be made as to the relative CPU performance of the algorithms compared in this section, as differences in computer processor speed and differences as to what is actually included in the time measurements still prevent any concrete conclusions being drawn. The more complete documentation of computer processor performance presented in this study should allow future researchers to make more meaningful comparisons.

4 Discussion

To gain a deeper understanding of the runtime behavior of PBS and the efficacy of its underlying mechanisms, additional empirical analyses were performed. Specifically, the variability in runtime between multiple independent runs of PBS on the same problem instance; the effect of adding additional computer processors; the efficiency of the genetic operators; and the effectiveness of the local search were investigated.

Table 4: PBS-1 Results, from 100 Trials, All of Which Were Successful, and VNS (Mladenović et al., 2003) Results, for the Class *ORLIB*: Target Cost for PBS-1 is That Reported as Average Cost for VNS; Avg. CPU Is Average CPU Time in Seconds, *SD* Is the Standard Deviation of the CPU Times. CPU and SD Values < 0.01 Are Shown as $< \epsilon$

			Target	VNS	PBS-1					Target	VNS	PBS-1	
File	n	p	cost	CPU	CPU	SD	File	n	p	cost	CPU	CPU	SD
pmed01	100	5	127	0.04	$< \epsilon$	$< \epsilon$	pmed21	500	5	40	1.24	0.02	0.01
pmed02	100	10	98	4.45	0.01	0.01	pmed22	500	10	38	82.63	1.23	1.31
pmed03	100	10	93	0.03	0.06	0.06	pmed23	500	50	23	112.33	0.02	0.01
pmed04	100	20	74	0.30	$< \epsilon$	0.01	pmed24	500	100	16	264.46	0.02	0.01
pmed05	100	33	48	0.11	$< \epsilon$	$< \epsilon$	pmed25	500	167	12	175.67	0.01	0.01
pmed06	200	5	84	1.51	0.02	0.02	pmed26	600	5	38	0.58	0.04	0.03
pmed07	200	10	64	1.30	0.01	0.02	pmed27	600	10	32	5.07	0.04	0.04
pmed08	200	20	55	2.22	0.01	0.02	pmed28	600	60	19	25.09	0.02	0.01
pmed09	200	40	37	10.89	$< \epsilon$	$< \epsilon$	pmed29	600	120	13	762.44	0.05	0.03
pmed10	200	67	20	9.04	$< \epsilon$	$< \epsilon$	pmed30	600	200	11	196.95	0.02	$< \epsilon$
pmed11	300	5	59	1.52	0.04	0.06	pmed31	700	5	30	0.58	0.03	0.01
pmed12	300	10	51	5.39	0.01	0.02	pmed32	700	10	29	165.26	0.32	0.45
pmed13	300	30	36	10.21	0.05	0.05	pmed33	700	70	16	806.77	0.04	0.01
pmed14	300	60	26	70.67	0.01	0.01	pmed34	700	140	12	160.15	0.03	$< \epsilon$
pmed15	300	100	18	55.86	$< \epsilon$	$< \epsilon$	pmed35	800	5	30	6.67	0.10	0.12
pmed16	400	5	47	0.08	0.01	0.01	pmed36	800	10	27	105.99	1.02	1.04
pmed17	400	10	39	26.00	0.02	0.02	pmed37	800	80	16	1197.86	0.04	0.01
pmed18	400	28	28	119.61	0.14	0.12	pmed38	900	5	29	1.92	0.03	0.01
pmed19	400	80	19	300.11	0.01	$< \epsilon$	pmed39	900	10	24	5.98	0.05	0.03
pmed20	400	133	14	218.61	0.01	$< \epsilon$	pmed40	900	90	14	493.79	0.40	0.25
							Total				5409.38	3.94	

Table 5: CPU Averages for PBS-1 (from 100 Trials, All of Which Were Successful) and VNS (Mladenović et al., 2003) for the *TSPLIB* Instances *u1060* and *pcb3038*. Target Cost for PBS-1 Is That Reported as Average Cost for VNS. Avg. CPU is Average CPU Time in Seconds, *SD* Is the Standard Deviation of the CPU Times. The %Dev. Column Shows the Average PBS Percentage Deviation from the PBS Target Cost

		Target	VNS	PBS-1					Target	VNS	PBS-1		
File	p	cost	CPU	CPU	SD	%Dev	File	p	cost	CPU	CPU	SD	%Dev
u1060	10	2280.09	94.93	7.89	5.29	0.00	pcb3038	50	317.00	578.81	66.10	69.93	−0.27
u1060	20	1611.95	20.49	21.79	13.71	−3.19	pcb3038	100	220.06	570.60	61.03	46.25	−0.05
u1060	30	1220.41	373.46	9.25	9.87	−1.28	pcb3038	150	174.83	52.99	460.53	289.34	−0.05
u1060	40	150.45	279.75	20.64	31.50	−3.55	pcb3038	200	157.88	747.00	9.40	9.82	−0.14
u1060	50	922.14	477.18	10.46	14.36	−0.07	pcb3038	250	140.98	103.72	4.30	3.71	−0.07
u1060	60	806.52	446.89	4.30	5.23	−0.17	pcb3038	300	123.33	786.96	6.01	3.16	−0.03
u1060	70	721.37	422.73	16.86	17.14	0.00	pcb3038	350	118.02	264.59	1.88	1.51	0.00
u1060	80	670.53	398.84	18.10	23.84	−1.45	pcb3038	400	107.65	904.88	5.87	2.53	−0.08
u1060	90	640.23	111.08	3.34	2.10	−0.01	pcb3038	450	101.51	674.55	3.16	1.31	−0.11
u1060	100	582.92	430.33	1.96	1.72	0.00	pcb3038	500	94.37	937.70	4.07	2.96	−0.01
u1060	110	565.72	186.60	1.47	1.00	−0.01	**Total**			5621.80	622.35		
u1060	120	551.90	218.84	0.83	0.70	0.00							
u1060	130	500.14	473.65	5.60	5.13	0.00							
u1060	140	500.12	215.06	0.24	0.21	0.00							
u1060	150	453.16	428.16	1.02	0.80	0.00							
Total			4577.99	123.75									

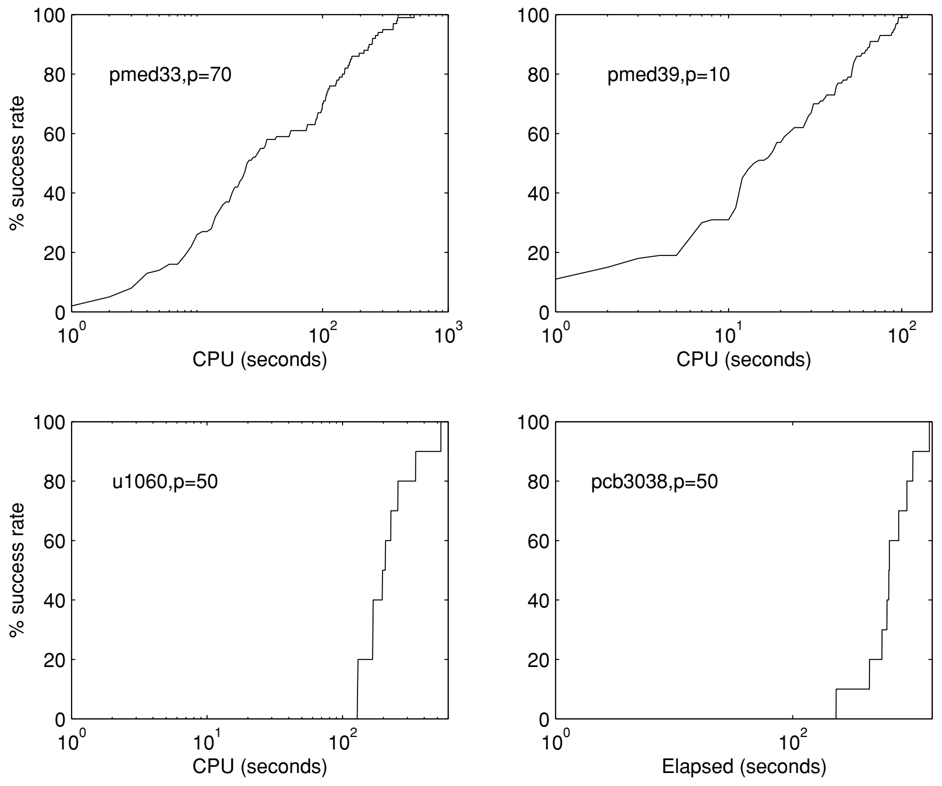

Figure 1: Runtime distributions of CPU time for PBS-1 for 100 trials (all successful) on the *pmed33*, $p = 70$, target p-center cost $= 15$, *pmed39*, $p = 10$, target p-center cost $= 23$, and 10 trials (all successful) for both *u1060*, $p = 50$ and *pcb3038*, $p = 50$.

4.1 Variability in Runtime

The variability of runtime between multiple independent trials of PBS for a particular instance is an important aspect of the behavior of stochastic algorithms. Following the methodology of Hoos and Stützle (2004), this aspect is presented based on runtime distributions (RTDs) of PBS-1 on four problem instances. As can be seen from the empirical RTD graphs shown in Figure 1, PBS-1 shows a reasonably large variability in runtime which is consistent with similar results for other high-performance stochastic search algorithms, for example, for maximum clique (Pullan and Hoos, 2006) and scheduling problems (Watson et al., 2005). Similar observations were made for PBS with regard to the other PC benchmark instances used in this study.

4.2 Computer Processor Scalability of PBS

The effect of adding computer processors for PBS to utilize is shown in Table 6 where the median elapsed time for 10 consecutive successful trials for the instance *u1060*, $p = 50$ and target p-center cost $= 904.92$ is shown. For PBS-x, all $\mathcal{C}$, $\mathcal{X}_1$, $\mathcal{X}_2$, $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{L}$ operations are performed in parallel in the slave tasks thus providing an x-way overlapping of the mutation, crossover, and local search portions of the PBS algorithm.

Table 6: Effect of the Number of Processors on the PBS-x Median Elapsed Time, Over 10 Successful Trials, for the Instance *u1060* with $p = 50$, Target p-center Cost = 904.92

x (Number of slave processors)	Median elapsed time
1	196.00
2	101.00
3	52.00
4	16.00

Table 7: Percentage Frequency of the Local Search, During the Initial Creation of P and After Each Genetic Operator, Obtaining the Target p-center Cost for All Instances of the *ORLIB*, *d493*, *d657*, *u1060*, *rl1323*, and *u1817* Classes

	ORLIB	*d493*	*d657*	*u1060*	*rl1323*	*u1817*
$\mathcal{L}$ Invocation	%Freq.	%Freq.	%Freq.	%Freq.	%Freq.	%Freq.
During creation of P	98.9	71.9	52.8	42.5	4.3	39.3
$\mathcal{M}_1$	0.1	3.0	2.8	6.4	7.4	3.3
$\mathcal{M}_2$ (after $\mathcal{M}_1$)	0.2	4.3	10.9	2.8	4.3	4.7
$\mathcal{M}_2$ (after $\mathcal{X}_1$)	0.2	4.3	5.0	6.4	8.5	6.0
$\mathcal{M}_2$ (after $\mathcal{X}_2$)	0.4	4.5	17.7	12.8	17.0	24.0
$\mathcal{X}_1$ (before $\mathcal{M}_2$)	0.1	4.5	2.8	7.8	11.7	8.0
$\mathcal{X}_2$ (before $\mathcal{M}_2$)	0.1	7.5	8.0	21.3	46.8	14.7

4.3 Effectiveness of PBS Components

Table 7 gives the frequency with which the local search ($\mathcal{L}$) located the target cost after being invoked during the initial creation of the population P and after each of the genetic operators for all instances of the *ORLIB*, *d493*, *d657*, *u1060*, *rl1323*, and *u1817* classes. The effectiveness of the directed mutation ($\mathcal{M}_2$) and phenotype ($\mathcal{X}_2$) genetic operators, as compared to the random versions, for the more difficult classes is highlighted by the results shown in Table 7 where, for the *u1060* class, $\mathcal{M}_2$ and $\mathcal{X}_2$ accounted for 43.3% of target cost locations (as compared to a total of 14.2% for $\mathcal{M}_1$ and $\mathcal{X}_1$), for *rl1323*, 76.6% of target cost locations (19.1% for $\mathcal{M}_1$ and $\mathcal{X}_1$) and for *u1817*, 49.4% of target cost locations (11.3% for $\mathcal{M}_1$ and $\mathcal{X}_1$).

Figure 2 gives more detailed information on the overall performance of the two genetic crossover operators in that it shows the evolution, for each application of each crossover operator, of the average p-center cost for the parents to the actual p-center cost of the child solutions after the genetic operator but prior to the local search. The effectiveness of the genetic operators in generating a wide range of starting points for the local search, from a population of eight solutions, is clearly demonstrated by the wide range of child p-center costs. The horizontal bands in the plots are caused by repeated use of the same parent p-center solutions from P. Of interest is that $\mathcal{X}_2$, on average, generates child p-center solutions of lower cost than those generated by $\mathcal{X}_1$ which illustrates the effectiveness of phenotype crossover operators for problems such as p-center.

4.4 Local Search

Figures 3 and 4 show, for the *pmed23*, *pmed33*, *u1060*, and *u1817* instances, the average cost of the p-center solution at each iteration of the PBS local search. In all cases, there is an initial fast decrease followed by an extended search over a relatively flat

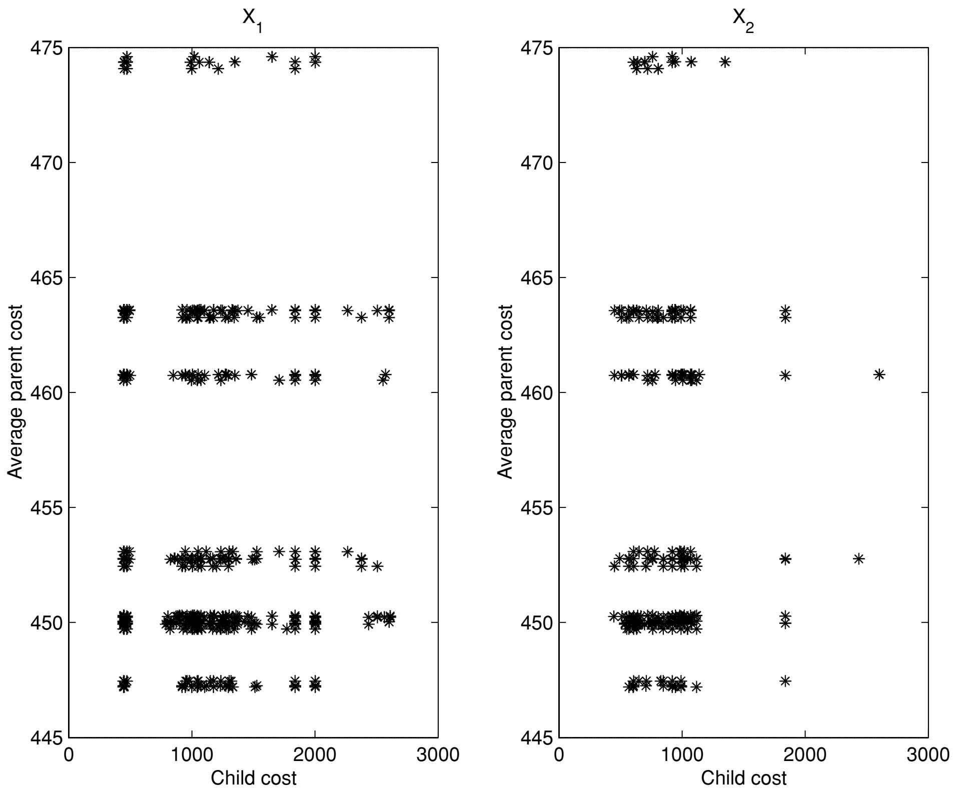

Figure 2: For the *u1060* instance, cost (prior to the local search) of the child p-center solutions generated by $\mathcal{X}_1$ and $\mathcal{X}_2$ as compared to the average parent cost.

cost hypersurface. For *pmed23* and *pmed33*, as p-center cost is an integer value, a large proportion of the cost hypersurface appears to consist of flat areas with ± 1 local maxima/minima. For *u1060* and *u1817*, p-center cost is a real value which gives rise to a cost hypersurface with few actual flat areas but a very large number of local minima. Clearly, the acceptance of non-improving facility-vertex swaps and preventing the re-use of facility-vertex swaps within a single invocation of local search are important features of the PBS local search in dealing with these types of hypersurfaces.

Also shown in Figures 3 and 4 is the actual progress of local search for the successful invocation of local search which, in all cases, started from a relatively low cost p-center solution. This indicates that the PBS meta-heuristic of generating new starting points for local search based on existing low cost p-center solutions (via the $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{X}_1$, $\mathcal{X}_2$ genetic operators) is an effective approach for the p-center problem.

Finally, Figures 3 and 4 show the percentage of invocations that reached each iteration level of local search. The effect of the local search termination condition $0.1(g+1)n$ in ensuring that a reasonable search around the starting point is performed before termination is evident.

5 Conclusions and Future Work

This study has demonstrated that by applying the general paradigm of using local search, within a population based meta-heuristic incorporating directed mutation and

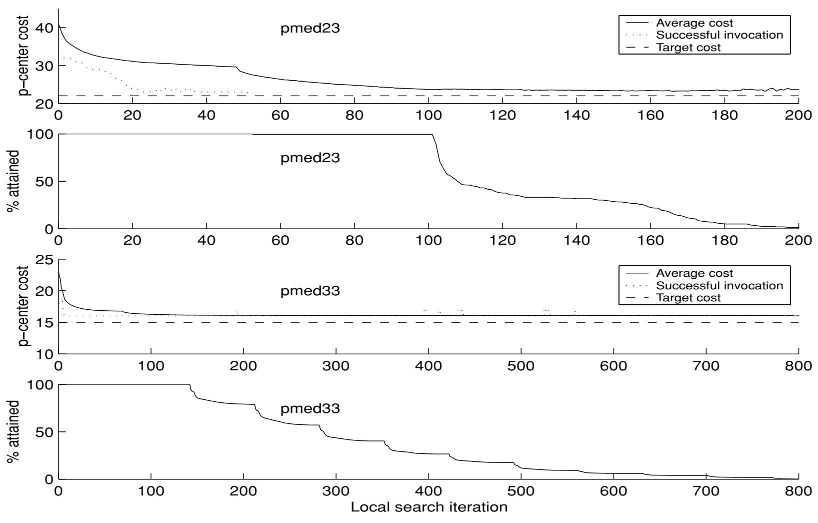

Figure 3: For the *pmed23* and *pmed33* instances, average cost of the p-center solution at each iteration of the PBS local search. Also shown are the actual progress of local search for the successful invocations which, in both cases, started from a relatively low cost p-center solution and the frequency with which each invocation of local search reached a given number of iterations.

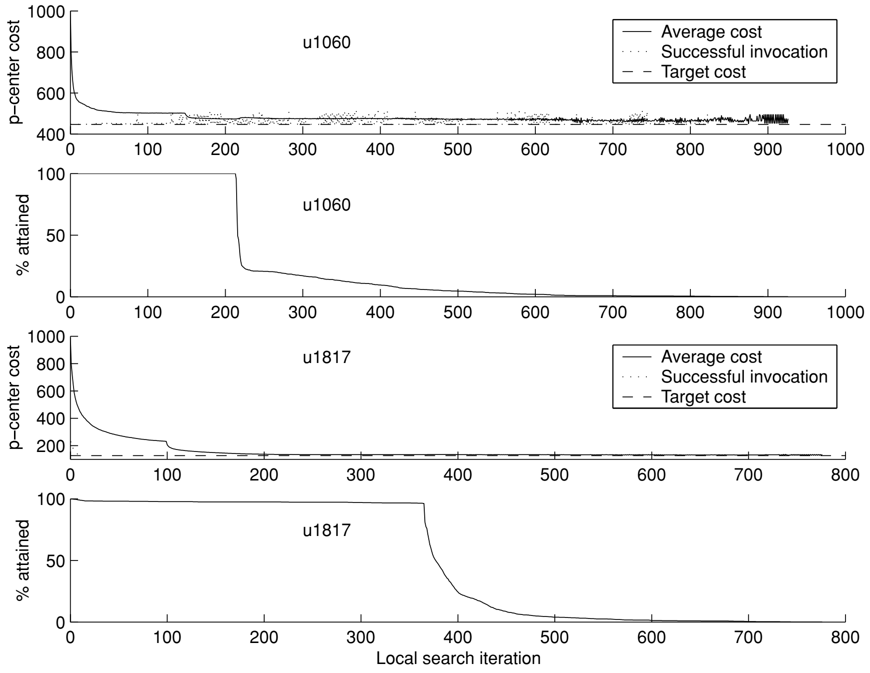

Figure 4: For the *u1060* and *u1817* instances, average cost of the p-center solution at each iteration of the PBS local search. Also shown are the actual progress of local search for the successful invocations which, in both cases, started from a relatively low cost p-center solution and the frequency with which each invocation of local search reached a given number of iterations.

phenotype crossover genetic operators an effective meta-heuristic for the p-center problem can be created. In summary, with regard to the robustness and quality of the PBS results, for the 138 benchmark instances, for the 116 instances where the optimal solution is known, PBS located 112 of these with a success rate of 100%. For the 22 instances where the optimal solution is not known, PBS located 19 new best known solutions and two currently best known solutions. With regard to computational performance, while the comparisons performed with the only other relevant meta-heuristic suggest that PBS has excellent performance, due to difficulties in realistically comparing computational performance with that documented for other algorithms, no final conclusive statement can be made as to the relative CPU performance of PBS as compared to other algorithms.

Given the excellent performance of PBS on the standard PC instances reported here, the algorithm clearly has substantial potential to provide the basis for high-performance algorithms for other clustering combinatorial optimization problems such as capacitated p-median, maximum covering, uncapacitated/capacitated facility location, set covering, and weighted maximum satisfiability.

References

Al-Khedhairi, A. and Salhi, S. (2005). Enhancements to two exact algorithms for solving the vertex p-center problem. *Journal of Mathematical Modelling and Algorithms*, 4:129–147.

Beasley, J. E. (1985). A note on solving large p-median problems. *European Journal of Operational Research*, 21:270–273.

Cheng, T., Kang, L., and Ng, C. (2007). An improved algorithm for the p-center problem on interval graphs with unit lengths. *Computers and Operations Research*, 4(8):2215–2222.

Cranic, T., Gendreau, M., Hansen, P., and Mladenović, N. (2004). Cooperative parallel variable neighbourhood search for the p-median. *Journal of Heuristics*, 10:293–314.

Daskin, M. (1995). *Network and Discrete Locations: Models, Algorithms and Applications*. Wiley, New York.

Daskin, M. (2000). A new approach to solving the vertex p-center problem to optimality: Algorithm and computational results. *Communications of the Operations Research Society of Japan*, 45:428–436.

Elloumi, S., Labbe, M., and Pochet, Y. (2004). A new formulation and resolution method for the p-center problem. *INFORMS Journal on Computing*, 16:84–94.

Hansen, P., Mladenović, N., and Perez-Brito, D. (2001). Variable neighborhood decomposition search. *Journal of Heuristics*, 7:335–350.

Hassin, R., Levin, A., and Morad, D. (2003). Lexicographic local search and the p-center problem. *European Journal of Operational Research*, 151:265–279.

Hochbaum, D. and Shmoys, D. (1985). A best possible heuristic for the k-center problem. *Mathematical Operations Research*, 10:180–184.

Hoos, H. H. and Stützle, T. (2004). *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, San Mateo, CA.

Ilhan, T. and Pınar, M. C. (2001). An efficient exact algorithm for the vertex p-center problem. Online report, `http://www.optimization-online.org/db_html/2001/09/376.html`.

Kariv, O. and Hakimi, L. (1979). An algorithmic approach to nework location problems, Part ii, The p-medians. *SIAM Journal of Applied Mathematics*, 37:539–560.

Krumke, S. (1995). On a generalization of the p-center problem. *Information Processing Letters*, 56(2):67–71.

Martinich, J. (1988). A vertex-closing approach to the p-center problem. *Naval Res. Log.*, 35:185–201.

Minieka, E. (1970). The m-center problem. *SIAM Review*, 12:138–139.

Mladenović, N., Labbe, M., and Hansen, P. (2003). Solving the p-center problem with tabu search and variable neighbourhood search. *Networks*, 42:48–64.

Plesnik, J. (1987). A heuristic for the p-center problem in graphs. *Discrete Applied Mathematics*, 17:263–268.

Pullan, W. (2005). An unbiased population-based search for the geometry optimization of Lennard-Jones clusters: $2 \leq n \leq 372$. *Journal of Computational Chemistry*, 26:899–906.

Pullan, W. and Hoos, H. (2006). Dynamic local search for the maximum clique problem. *Journal of Artificial Intelligence Research*, 26:159–185.

Reinelt, G. (1991). TSPLIB: A traveling salesman problem library. *ORSA Journal on Computing*, 3:376–384.

Resende, M. and Werneck, R. (2004). A hybrid heuristic for the p-median problem. *Journal of Heuristics*, 10:59–88.

Scaparra, M., Pallottino, S., and Scutellà, M. (2004). Large-scale local search heuristics for the capacitated vertex p-center problem. *Networks*, 43(4):241–255.

Taillard, E. D. (2003). Heuristic methods for large centroid clustering problems. *Journal of Heuristics*, 9:51–74.

Tansel, B. C., Francis, R. L., and Lowe, T. J. (1983). Location on networks: A survey. *Management Science*, 29:482–511.

Watson, J., Whitley, L., and Howe, A. (2005). Linking search space structure, run-time dynamics, and problem difficulty: A step toward demystifying tabu search. *Journal of Artificial Intelligence*, 24:221–261.

Whitaker, R. (1983). A fast algorithm for the greedy interchange of large-scale clustering and median location problems. *INFOR*, 21:95–108.