INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Asynchronous Cooperative Local Search for the Office-Space-Allocation Problem

Dario Landa-Silva, Edmund K. Burke,

To cite this article:
Dario Landa-Silva, Edmund K. Burke, (2007) Asynchronous Cooperative Local Search for the Office-Space-Allocation Problem. INFORMS Journal on Computing 19(4):575-587. https://doi.org/10.1287/ijoc.1060.0200

INFORMS Journal on Computing
Vol. 19, No. 4, Fall 2007, pp. 575–587
ISSN 1091-9856 | EISSN 1526-5528 | 07 | 1904 | 0575

DOI 10.1287/ijoc.1060.0200

Asynchronous Cooperative Local Search for the Office-Space-Allocation Problem

Dario Landa-Silva, Edmund K. Burke
Automated Scheduling, Optimisation and Planning Research Group, School of Computer Science,
University of Nottingham, Nottingham NG8 1BB, United Kingdom {jds@cs.nott.ac.uk, ekb@cs.nott.ac.uk}

We investigate cooperative local search to improve upon known results of the office-space-allocation problem in universities and other organizations. A number of entities (e.g., research students, staff, etc.) must be allocated into a set of rooms so that the physical space is utilized as efficiently as possible while satisfying a number of hard and soft constraints. We develop an *asynchronous cooperative local search* approach in which a population of local search threads cooperate asynchronously to find better solutions. The approach incorporates a cooperation mechanism in which a pool of genes (parts of solutions) is shared to improve the global search strategy. Our implementation is single-processor and we show that asynchronous cooperative search is also advantageous in this case. We illustrate this by extending four single-solution metaheuristics (hill-climbing, simulated annealing, tabu search, and a hybrid metaheuristic) to population-based variants using our asynchronous cooperative mechanism. In each case, the population-based approach performs better than the single-solution one using comparable computation time. The asynchronous cooperative metaheuristics developed here improve upon known results for a number of test instances.

Key words: artificial intelligence; heuristics; approximation algorithms; office-space-allocation
History: Accepted by Michel Gendreau, Area Editor for Heuristic Search and Learning; received August 2004; revised July 2005, June 2006; accepted July 2006. Published online in *Articles in Advance* August 20, 2007.

1. Introduction

We tackle the office-space-allocation problem, which arises in universities and other organizations when distributing office space among staff and other entities. This is a difficult, highly constrained combinatorial optimization problem. Recombination of solutions is not straightforward because it is very difficult to maintain feasibility of solutions when applying recombinative operators (Burke et al. 2001b). This difficulty motivates our interest in developing hybrid and cooperative metaheuristics.

We investigate asynchronous cooperative search to improve upon known results by developing population-based metaheuristics under the principle of *asynchronous cooperative local search*. Researchers have investigated synchronous and asynchronous cooperative search in the context of parallel metaheuristics (Le Bouthillier and Crainic 2005, Crainic 2005, Crainic and Toulouse 2003, Crainic et al. 2005). Our implementation has two distinctive aspects. First, it is a sequential single-processor implementation where a population of local search threads are allowed to carry out their individual search and also cooperate in an asynchronous manner to improve the global search. Secondly, instead of sharing complete solutions between the threads (as in Le Bouthillier and Crainic 2005) our approach builds a global memory using "parts" of solutions collected in a pool shared by all the local search threads. This pool of genes contains parts of solutions, some of them regarded as "good" and others regarded as "bad." The population of threads uses this shared information in two ways: to use parts of solutions marked as "good" to improve their individual solution, and to avoid using parts of solutions marked as "bad" when reconstructing solutions after a disruption operator is applied as a diversification strategy. By having a population of local searchers that share the information obtained during the search, a form of recombination can be achieved and performance of the local search can be improved. We extend a range of single-solution local search algorithms toward population-based metaheuristics using our asynchronous cooperative mechanism. We carry out experiments to compare performance of the original and extended variants of the algorithms when applied to several test instances.

Section 2 describes the problem domain and test instances, while Section 3 presents the heuristics implemented. Section 4 describes in more detail our *asynchronous cooperative local search* approach. Section 5 gives details of our experiments and presents and discusses the results. Section 6 gives final remarks and proposes future research.

2. The Office-Space-Allocation Problem

2.1. The Problem Domain

The office-space-allocation problem in universities and other organizations refers to the distribution of rooms among people and other entities such as laboratories, lecture rooms, etc. (Burke and Varley 1998, Landa Silva 2003). The goal is to find an allocation that optimizes space utilisation, satisfies all hard constraints, and satisfies as many soft constraints as possible. Constraints restrict the relative position and grouping of entities. There are proximity/adjacency, grouping, and sharing constraints. Proximity/adjacency constraints exist when one entity should be close to certain rooms or to other entities. Grouping constraints refer to a group of entities that should be allocated close to each other. Sharing constraints indicate that entities should not share a room. An additional goal is very difficult to accomplish and to evaluate: satisfaction of each person with the assigned room. Decision makers (space administrators) usually require various high-quality solutions so they can select the most appropriate one (Burke and Varley 1998).

2.2. Problem Formulation

This is similar to other combinatorial optimization problems such as bin packing, the knapsack problem, the general assignment problem, and their variants (Martello and Toth 1990). However, in our problem the capacity of rooms can be exceeded (but penalized). Also, there are (hard and soft) constraints that affect the relative position of entities in the allocation. We formulate the problem as follows:

$n =$ number of entities with sizes $s_1, s_2, \ldots, s_n$.

$m =$ number of rooms with capacities $c_1, c_2, \ldots, c_m$.

$x_{ij} = 1$ if entity j is assigned to room i, and 0 otherwise.

x is the matrix of $[x_{ij}]$ values.

$h =$ number of hard constraints of the form $Z(r) =$ *true*.

$g =$ number of soft constraints of the form $Z(r) =$ *true*.

The problem is to minimize over x

$$F(x) = f_1(x) + f_2(x) \tag{1}$$

subject to

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

$$Z(r) = true \quad \text{for } r = 1, 2, \ldots, h,$$

where the space-misuse function $f_1(x)$ and violation-of-soft-constraints function $f_2(x)$ are

$$f_1(x) = \sum_{i=1}^{m} WP_i + \sum_{i=1}^{m} OP_i$$

$$f_2(x) = \sum_{r=1}^{g} SCP_r.$$

SCP_r is the penalty for violating the r^{th} soft constraint. The constraint types (soft and hard) and associated penalties for the problem instances used here are in Section 2.3. The amount of capacity wasted or overused for each room i is given respectively by

$$WP_i = \max\left(0, c_i - \sum_{j=1}^{n} x_{ij} s_j\right)$$

$$OP_i = \max\left(0, 2\left(\sum_{j=1}^{n} x_{ij} s_j - c_i\right)\right). \tag{2}$$

Note that for each room i only one of WP_i or OP_i has a value greater than zero depending on whether the room's capacity is wasted or overused. Exceeding the room's capacity is considered worse than wasting it and therefore a multiplication factor of 2 is applied when calculating OP_i.

An allocation or solution is an n-dimensional vector $Y = [y_1, y_2, \ldots, y_n]$ where the jth ($j = 1, \ldots, n$) position in the vector corresponds to the jth entity and the value $y_j = i$ ($i = 1, \ldots, m$) means that the jth entity has been allocated to the ith room. That is, if the decision variable $x_{35} = 1$ then entity $j = 5$ has been allocated to the area of space $i = 3$ and hence, $y_5 = 3$ in the n-dimensional vector Y. The nonsimilarity or diversity in the solution space between two allocations is evaluated as follows. The two allocations $Y' = [y'_1, y'_2, \ldots, y'_n]$ and $Y'' = [y''_1, y''_2, \ldots, y''_n]$ are considered to be completely nonsimilar if $y'_j \neq y''_j$ ($j = 1, \ldots, n$), i.e., if each of the n entities has been allocated to a different room in each solution. The diversity of a population of solutions is (3), where d_j is the number of different values in the jth position for all vectors and p is the population size.

$$V_p = \frac{\sum_{j=1}^{n}((d_j - 1)/(p - 1))}{n} 100 \tag{3}$$

A population of solutions using different rooms to allocate each different entity has a diversity $V_p = 100\%$. If all solutions in the population use the same room to allocate each entity, then $V_p = 0\%$. Our interest is to obtain a set of solutions that are diverse in the solution space (high V_p) while their fitness (1) should be of similar high quality (low $F(x)$). This is because the decision makers are usually more interested in having a set of solutions that represent different allocations.

Table 1(a) Description of Test Instances

	nott1 $n=158$ $m=131$		nott1b $n=104$ $m=77$		nott1c $n=94$ $m=94$		trent1 $n=151$ $m=73$	
Constraints	Hard	Soft	Hard	Soft	Hard	Soft	Hard	Soft
Not sharing	100	58	46	58	84	10	80	71
Be allocated in	0	35	0	9	0	35	0	19
Be adjacent to	5	15	4	10	5	15	0	5
Be away from	6	14	1	2	5	12	0	0
Be together with	0	20	0	20	0	0	0	36
Be grouped with	0	10	0	9	0	10	0	0
Total	263		159		176		211	

Table 1(b) Brief Description of Each Type of Constraint and Penalty Applied

Constraints	Penalty	Description
Not sharing	50	Entity not to share a room
Be allocated in	20	Entity allocated in a specific room
Be adjacent to	10	Entity adjacent to another entity
Be away from	10	Entity away from another entity or room
Be together with	10	Entities allocated in the same room
Be grouped with	5	Entities allocated close to each other

2.3. Test Instances

Four test instances, prepared using real data supplied by British universities, are used here; see Table 1(a). A brief description of each type of constraint and the penalty applied when one constraint of each type is violated are also given; see Table 1(b). The nott1 instance is the largest and most constrained. These and other data sets are available in the Online Supplement to this paper on the journal's website.

3. Solving the Problem

3.1. Previous Work

Giannikos et al. (1995) applied goal programming to distribute offices among staff while Ritzman et al. (1980) used linear programming to automate the assignment of 144 offices to 289 staff. Benjamin et al. (1992) also applied linear programming to plan the layout of a computer-integrated manufacturing laboratory to allocate space. These studies recognized that it is virtually impossible to allocate space to satisfy all constraints.

We investigated genetic algorithms in Burke et al. (2001b). Despite designing specialized genetic operators to deal with the existing constraints, local search performed better. Components from various metaheuristics were incorporated into one single-solution hybrid approach with very good results (Burke et al. 2001a), and extended toward population-based variants via an annealing schedule to control evolution of the population. We also investigated population-based approaches as a biobjective optimization (Burke et al. 2001c, Burke and Landa Silva 2006). For more detail see Landa Silva (2003).

3.2. Initialization and Neighborhood Search

The following peckish (not-so-greedy) constructive heuristic H_I is used to initialize solutions by allocating one entity at a time while ensuring satisfaction of hard constraints. We set $k = n/3$ by preliminary experimentation.

H_I. Heuristic for initialization of solutions

Step 1. Select an unallocated entity e_j uniformly at random.

Step 2. Select $k < m$ uniformly at random rooms with adequate remaining space c_i^* to allocate e_j; c_i^* should be between $0.5s_j$ and $1.5s_j$ where s_j is the size of entity e_j.

Step 3. Select the best of the k rooms to allocate entity e_j, taking into consideration the minimization of space misuse $f_1(x)$ and the satisfaction of hard and soft constraints $f_2(x)$.

Step 4. Stop if all entities are allocated; otherwise go to Step 1.

The neighborhood search for feasible candidate solutions (all entities allocated and all hard constraints satisfied) uses three moves:

1. *swap* the allocation between two entities, i.e., swap values between two positions in Y, the n-dimensional vector representing the solution;

2. *interchange* the allocated entities between two rooms, i.e., given two values y_{j1} and y_{j2} for $j1, j2 \in \{1, 2, \ldots, m\}$ in Y, change to y_{j2} the value of all positions having the value y_{j1} and change to y_{j1} the value of all positions having the value y_{j2};

3. *relocate* or change the room assigned to an entity, i.e., change the value of a given position in Y to a different value within $\{1, 2, \ldots, m\}$.

The heuristic H_{LS} below is used for neighborhood search. This is a best-of-($n/3$) improvement neighborhood search (i.e., $k = n/3$). Other variations such as first improvement and best-of-all improvement were also tried but were less successful.

H_{LS}. Heuristic for neighborhood search

Step 1. Select one type of move (*swap*, *interchange*, or *relocate*) uniformly at random.

Step 2. Generate and evaluate k neighboring feasible solutions using the type of move selected in Step 1.

Step 3. Select the best of the k candidate solutions generated in Step 2.

3.3. Iterative Improvement

After an initial solution is constructed with H_I, the iterative improvement algorithm performs neighborhood search using heuristic H_{LS} until a termination criterion is satisfied. The pseudocode for this approach is given below. Note that Step 3.1 is included here to extend this single-solution approach to a population-based variant below. The *asynchronous cooperation mechanism* is described later.

Iterative improvement algorithm

Step 1. Generate the current solution x using heuristic H_I.

Step 2. Find a feasible candidate solution x' using heuristic H_{LS} and if x' is found,

2.1. Evaluate x' and if it is better than x then $x \leftarrow x'$.

Step 3. If no feasible solution x' was found,

3.1. If in cooperative scheme then use the *asynchronous cooperation mechanism*.

Step 4. Stop if the termination condition is satisfied; otherwise go to Step 2.

3.4. Simulated Annealing

Simulated annealing attempts to avoid getting stuck in poor local optima by accepting nonimproving candidate solutions (Aarts et al. 2005). The pseudocode for our implementation of this approach is given below. Step 4.1 is included here for extending this approach to a population-based variant below.

Simulated-annealing algorithm

Step 1. Generate the currrent solution x using heuristic H_I.

Step 2. Set $T_0 \leftarrow 1{,}000$, $T \leftarrow T_0$, $\Delta T \leftarrow 200$, $R_{iter} \leftarrow 0$, $R_{step} \leftarrow 10n$, $D_{step} \leftarrow n/2$, and $iter \leftarrow 0$.

Step 3. Find a feasible candidate solution x' using heuristic H_{LS} and if x' is found,

3.1. Calculate the fitness variation ΔF between x and x'.

3.2. If x' is better than x then $x \leftarrow x'$.

3.3. If x' is not better than x and $T = 0$,

3.3.1. Set $R_{iter} \leftarrow R_{iter} + 1$ and if $(R_{iter} \bmod R_{step}) = 0$ set $T \leftarrow T_0$ and $R_{iter} \leftarrow 0$.

3.4. If x' is not better than x and $T > 0$,

3.4.1. Set *accept probability* $= e^{-\Delta F/T}$, if *accept probability* $>$ uniform $[0, 1]$ random number then $x \leftarrow x'$.

3.5. $iter \leftarrow iter + 1$.

3.6. If $(iter \mod D_{step}) = 0$ then $T \leftarrow T - \Delta T$.

3.7. If $T \leq 0$ then make $T \leftarrow T_0$.

Step 4. If no feasible candidate solution x' was found,

4.1. If in cooperative scheme then use the *asynchronous cooperation mechanism*.

Step 5. Stop if the termination criterion is satisfied; otherwise go to Step 3.

3.5. Tabu Search

Tabu search attempts to guide the search in a systematic and intelligent way by using adaptive and flexible memory structures and some intensification and diversification strategies (Glover and Laguna 1997, Gendreau and Potvin 2005). Our implementation is below. Step 4.1 is included here for extending this approach to a population-based variant below.

Tabu-search algorithm

Step 1. Generate the current solution x using heuristic H_I.

Step 2. Initialize the short-term and long-term memory structures (see below).

Step 3. Explore a set of candidate solutions by generating each candidate solution using heuristic H_{LS} and the memory structures. Select the best candidate solution and if a feasible candidate solution x' is found,

3.1. Evaluate x' and if it is better than x then $x \leftarrow x'$.

3.2. Update the short-term and long-term memory structures (see below).

Step 4. If no feasible solution x' was found,

4.1. If in cooperative scheme use the *asynchronous cooperation mechanism*.

Step 5. Stop if the termination criterion is satisfied; otherwise go to Step 3.

3.5.1. Memory Structures. Two $n \times m$ matrices are used and in both of them the cell (j, i) corresponds to the allocation of the jth entity to the ith room. The matrix M_T stores those pairs (entity, room) that will be considered as tabu for a number of iterations while the matrix M_A stores those pairs (entity, room) that will be considered attractive during the search. The tabu matrix M_T is updated each time a move suggested by the heuristic H_{LS} produces a detriment in the fitness of the current solution, while the attractive matrix M_A is updated each time the move produces an improvement. Updating a cell in M_T is done by

setting the value in the cell to *iter* + *tenure* so that a move involving the pair (entity, room) corresponding to that cell is set as tabu for *tenure* iterations. We use a $tenure = n$ and kept constant throughout all the iterations. Updating a cell in M_A is done by incrementing the value in the cell in one unit, i.e., $M_A(j, i) \leftarrow M_A(j, i) + 1$. In each type of move (*swap*, *interchange*, *relocate*), the cells that are updated are the ones corresponding to the pairs (entity, room) after implementing the move. For example, if the 6th entity is relocated from the 2nd to the 4th room and the move produced a better solution, then $M_A(6, 4)$ is incremented by one. If the move generated an inferior solution, then $M_T(6, 4)$ is set to *iter* + *tenure*. Note that in a *swap* move two cells are updated while in an *interchange* move more cells might be updated. The tabu (resp. attractive) matrix acts as the short-term (resp. long-term) memory component. Because both matrices store pairs (entity, room), this mechanism memorizes parts of solutions or genes that come from bad or good solutions.

3.5.2. Intensification and Diversification Strategies. M_T and M_A are used to implement intensification and diversification strategies. If the move found by H_{LS} is considered tabu according to M_T, i.e., any of the pairs (entity, room) in the implemented move is marked as tabu in M_T in the current iteration, then another move is sought unless the aspiration criterion is satisfied. The aspiration criterion is that the new candidate solution should be better than the best solution so far. The maximum number of attempts to find a feasible non-tabu move is set to $n/10$. If, after this, H_{LS} cannot find a move, a *relocate* move is heuristically created using the information stored in M_A. To do this, an entity j is selected uniformly at random and the highest value in the jth row is identified in M_A. This corresponds to the most attractive room i to allocate the entity j from the perspective of the current population and the history of the search. If the entity j is not already allocated to room i then the move proposed is to *relocate* the entity to that room. If this assignment already exists in the current solution or if the move creates an infeasible allocation, then another entity is selected uniformly at random and the same process in carried out for a maximum of $n/10$ attempts.

4. Asynchronous Cooperative Local Search

4.1. Related Work

There are a number of classification schemes describing evolutionary algorithms and their hybrids with local search (Hertz and Klober 2000, Calegari et al. 1999, Preux and Talbi 1999, Talbi 2002). Crainic et al. (1997) presented a taxonomy for parallel tabu search approaches in which the parallel approach is classified according to three features: the *control strategy* used to guide the search, the *information-sharing mechanism*, and the *strategies to partition the search space* to be explored by each thread. Similarly to Preux and Talbi (1999), Crainic et al. (1997) also noted that asynchronous parallel metaheuristics are promising and merit more research. Crainic and Toulouse (2003) surveyed parallel metaheuristics concentrating on simulated annealing, genetic algorithms, and tabu search; they noted that asynchronous cooperative multi-thread metaheuristics (like ours) appear to be less investigated but more approaches of this type are being proposed and that further theoretical and empirical research is required. Crainic et al. (2005) surveyed implementations of cooperative tabu search and identified asynchronous search as promising; they focused on the mechanisms for asynchronous cooperation in the literature and identified cooperative multi-thread methods and their hybrids as the current trend in cooperative heuristic search. In a vehicle routing problem, multiple search threads cooperating asynchronously and sharing a pool of solutions enhances the overall performance and robustness of the individual threads (Le Bouthillier and Crainic 2005). An earlier survey on parallel metaheuristics based on local search is Verhoeven and Aarts (1995).

4.2. The Cooperation Mechanism

The memory structures and intensification/diversification strategies described above for tabu search successfully direct the search toward high-quality neighbor solutions. These features are not part of the iterative-improvement and simulated-annealing algorithms, but we used the memory structures to develop a cooperation mechanism for *population-based asynchronous cooperative local search*. Later, the population-based variants of iterative improvement and simulated annealing will use the cooperation mechanism, but this is a feature of our asynchronous cooperative scheme and not of the local search threads. Having a population of local search threads, the matrices M_A and M_T are used as a global memory structure shared by the population. This strategy allows storing parts of attractive solutions in M_A and parts of unattractive solutions in M_T. The information stored in M_A and M_T is used in the *asynchronous cooperative local search* scheme in two ways:

1. **Information-Sharing Strategy**. Each explorer performs neighborhood search using heuristic H_{LS} but M_A and M_T are now updated by all individuals in the population. When a single-solution explorer cannot get a feasible solution from H_{LS}, i.e., when the *asynchronous cooperation mechanism* is invoked, a heuristic

is used to modify the solution using the information stored in M_A. This heuristic goes through each row j in M_A and explores the most attractive allocations for the jth entity. That is, it starts with the cell having the highest value and continues to the one with the lowest value and makes the allocation (entity, room) that keeps the solution feasible and is different from the one in the current solution. The changes are made even if the solution is worsened. A maximum of $n/20$ changes are implemented in this way.

2. **Diversification Strategy.** We implemented a strategy that "heavily" disrupts the current solution as follows. Those entities that are penalized the most (are involved in violation of soft constraints or in the misuse of space) are removed from their assigned rooms. Then, the strategy attempts to allocate each of these now-unallocated entities to one of various alternative rooms. For each unallocated entity, the rooms from the first one to the last one (i.e., $i = 1, \ldots, m$) are evaluated for a feasible allocation with the exception of those allocations marked as tabu in M_T. The degree of disturbance carried out by this diversification strategy is controlled by setting the maximum number of penalized entities that will be unallocated to be $n/5$. The purpose of this "heavy" disruption is to encourage each explorer to search a (hopefully) very different area of the solution space.

4.3. Extending the Single-Solution Approaches

A single-solution explorer (local searcher) LS_{SS} can be extended to a population-based approach LS_{PB} based on the *asynchronous cooperative mechanism* described above. The first phase (Step 1) in the pseudocode below corresponds to the construction of a population of explorers, each associated to an initial solution. In the intensification phase (Step 3.1) each explorer aims to achieve self-improvement using local search and the *information-sharing strategy*. In the diversification phase (Step 3.2), each explorer uniformly at random modifies its current solution using the *diversification strategy*. The best solution found by each explorer is maintained in the best population so far. Note that although the improvement rate could be better in some explorers, each explorer has its own solution and none is permitted to contribute more than one solution to the best population so far, i.e., this is a population of independent and cooperating threads (multi-walk approach). This encourages diversity and avoids one or more explorers to dominate the search.

LS_{PB}. Population-based scheme based on asynchronous cooperative local search

Step 1. Generate the initial current population P using heuristic H_I.

Step 2. Archive the current population as the best population so far, P^*.

Step 3. Each explorer carries out the following intensification and diversification steps asynchronously:

3.1. Do population self-improvement (intensification) updating P^*, i.e., each individual in P executes the single-solution local search approach LS_{SS} using the *information-sharing strategy* and attempts to improve its own solution iteratively. The local search thread continues doing this until no further self improvement to its current solution is possible.

3.2. Do uniform random variation of the individual solution (diversification), i.e., since the local search thread appears to be "stuck," its current solution is disturbed using the *diversification strategy*.

Step 4. Stop if the termination criterion is satisfied; otherwise continue with the process of Step 3.

Although we implemented our algorithms sequentially using a single-processor computer, the structure of these algorithms permits their parallel implementation. In our implementation, only one local search thread at a time can access the global memory structures M_A and M_T. If the asynchronous cooperative algorithms are parallelized, a mechanism should be implemented to resolve conflicts if two threads attempt to update the same cell in M_A or M_T at the same time. The value of the tabu *tenure* parameter in M_T is the same for all threads and fixed for the whole search process.

Parallel metaheuristics are expected to have a number of advantages over their nonparallel counterparts (Crainic 2005). They should be more robust, find better solutions, require little or no parameter tunning, while using comparable computing resources. Crainic (2005) also notes that exchanging "meaningful" information in a "timely" manner is an objective in multithreaded approaches but that current implementations tend to exchange complete solutions (the best ones) and clean up the memory structures before a new search begins. It is also believed that interrupting the local search does not always yields good results. Our approach addresses those issues by (1) sharing "good" and "bad" parts of solutions, (2) allowing each thread to access the global memory when needed (asynchronously), (3) updating the global memory along the search, and (4) allowing each thread to continue with its individual search until it gets stuck. Our experiments also show that the extended approaches achieve much better results and are more robust than the noncooperative independent approaches while using comparable computation time.

An extended population-based asynchronous cooperative approach consists of replacing LS_{SS} in the pseudocode above by a single-solution local search technique. The algorithms implemented here are the iterative improvement algorithm of Section 3.3 and its population-based variant (II_{SS} and II_{PB}), the

simulated-annealing algorithm of Section 3.4 and its population-based variant (SA_{SS} and SA_{PB}), and the tabu-search algorithm of Section 3.5 and its population-based variant (TS_{SS} and TS_{PB}). TS_{PB} maintains the global memory structures only, while in TS_{SS} the memory structures are local, as described in Section 3.5.

5. Performance of the Extended Approaches

5.1. The Experiments

One aim of our experiments is to evaluate how beneficial it is to extend a single-solution technique toward a population-based approach as proposed above. The other aim is to improve upon the known solutions. We seek high-quality solutions that are also diverse with respect to the solution space (see Section 2). The experiments are designed to compare the population-based variant against the corresponding single-solution technique for finding such set. A fair way to do that is to execute each method for an equal computation time. The number of solution evaluations or neighborhood moves is another possibility for comparison, but because the population-based approaches spend extra time using the *asynchronous cooperation mechanism*, this stopping criterion would be unfair for the single-solution methods. Preliminary experiments showed that, given a short computation time, the single-solution approaches quickly achieve improvement but they also get stuck relatively early. The population-based approaches can take longer (relative to the single-solution variants) to reach high-quality solutions. We carried out experiments to find the computation time for which the single-solution approaches achieved no further improvement for a considerable number of iterations.

Given an initial population P_I of size p, the single-solution approach was executed for t_{ind} computation time to each of the solutions in P_I and the best solution at the end of each run was archived (p solutions are obtained). Then, the corresponding population-based approach was executed for pt_{ind} computation time using P_I. This process was repeated ten times (a different P_I is generated each time) for each of the problem instances of Section 2.3. For each set of p solutions obtained, the best, average, and worst solution fitnesses were recorded and these values were averaged for each set of ten repetitions. To compare further the performance of each population-based variant against its corresponding single-solution algorithm, we carried out experiments using small and large populations with low and high diversity for each test instance as shown in Table 2. The algorithms were coded using MS Visual C++ 6.0 on a 700 MHz PC with 132 MB RAM.

5.2. Results on Fitness of Solutions

Figures 1–4 summarize the results using the various initial populations for each test instance. The left bar in each pair shows the results from the population-based variant of one algorithm, the right bar shows the results from the corresponding single-solution approach and a line is drawn between the average solutions. The solutions from the population-based variants are better than those from the single-solution approaches. The best, average, and worst solution qualities are better for the extended algorithms in most cases. This is clear for the nott1 and nott1c test instances in Figures 1 and 3, respectively. In the results for the nott1b instance in Figure 2, the extended simulated-annealing algorithm is outperformed by the single-solution approach when the initial population is small and the diversity is low (nott1bB2). In some cases the worst solution found by the population-based variant of one algorithm was worse than the one found by the corresponding single-solution approach. This is true for the simulated-annealing algorithm on the nott1bA, nott1bA2, nott1bB, and trent1A2 cases, and the tabu search algorithm on the nott1bB and nott1bB2 cases.

In some cases, even the worst solution produced by the extended algorithm outperforms, or at least matches, the quality of the best solution found by the corresponding single-solution approach. This is true for the iterative improvement algorithm on most cases of the nott1c and trent1 problems, the simulated annealing-algorithm on the nott1cA and trent1B cases, and tabu search on nott1cB, nott1cB2, and trent1A2. The size of each bar gives an indication of the difference in quality between the worst and best

Table 2 Initial Populations of Different Sizes and Diversity Values for Each Test Instance

	$p = 20$		$p = 5$	
	$65\% < V_{ip} < 90\%$	$20\% < V_{ip} < 40\%$	$65\% < V_{ip} < 90\%$	$20\% < V_{ip} < 40\%$
nott1, $t_{ind} = 120$ s	nott1A	nott1A2	nott1B	nott1B2
nott1b, $t_{ind} = 60$ s	nott1bA	nott1bA2	nott1bB	nott1bB2
nott1c, $t_{ind} = 30$ s	nott1cA	nott1cA2	nott1cB	nott1cB2
trent1, $t_{ind} = 70$ s	trent1A	trent1A2	trent1B	trent1B2

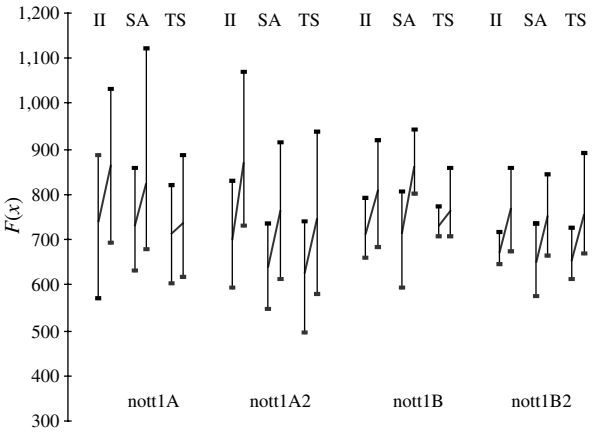

Figure 1 **Results for nott1**

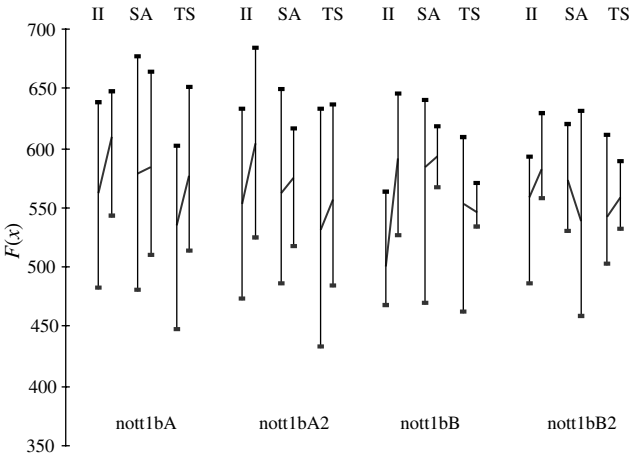

Figure 2 **Results for nott1b**

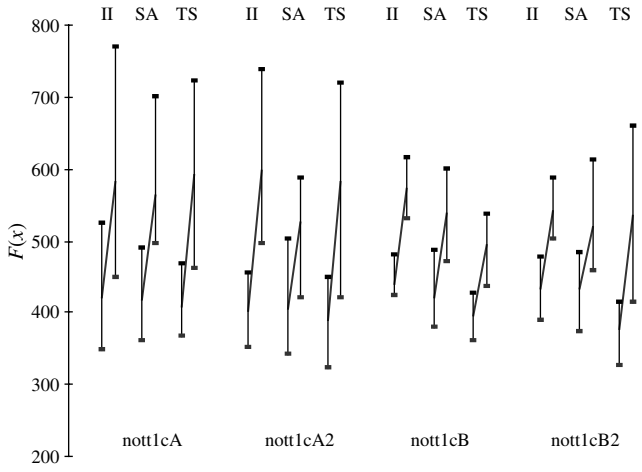

Figure 3 **Results for nott1c**

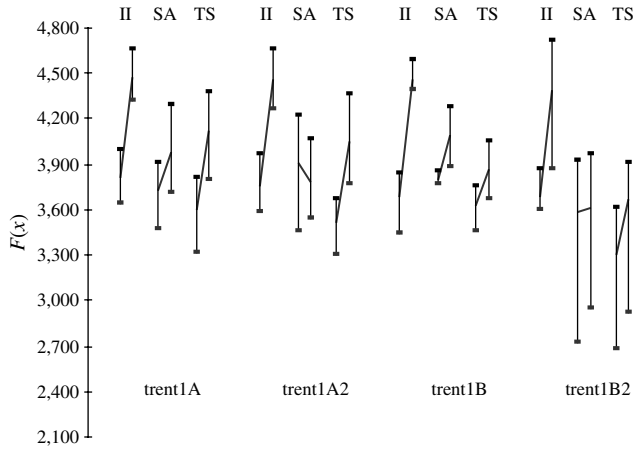

Figure 4 **Results for trent1**

solutions found by each algorithm variant. This difference appears smaller for the population-based approaches compared to the corresponding single-solution algorithms (except for nott1b). In some cases, the extended variant of a less-sophisticated algorithm outperforms the single-solution variant of another more elaborate technique; e.g., in Figure 1 the extended variant of iterative improvement clearly outperforms the single-solution variant of simulated annealing on nott1B.

5.3. Diversity of Solutions

Over the ten runs, we calculated the average final population diversity V_{fp} (3) for each algorithm variant. When the initial population is highly diverse ($65\% < V_{ip} < 90\%$), the population-based variants achieve a V_{fp} of around 10% below V_{ip}, while the corresponding single-solution approaches achieve a V_{fp} of around 15% below V_{ip} in most cases. In only a few cases the V_{fp} obtained by an extended approach is slightly lower than that of the corresponding single-solution variant. If the initial population has a low diversity ($20\% < V_{ip} < 40\%$), the extended approach typically obtains a V_{fp} of around 150% above V_{fp}. Although the single-solution variants also achieve improvement in this respect, typical values are around 50% above V_{fp}.

The population-based algorithms consistently maintain good population diversity (above 40%) even if the computation times are considerably longer (as in the experiments in Section 5.4). We attribute this to the way in which our *asynchronous cooperative mechanism* operates. Each local search thread focuses on self improvement and accesses the shared memory structures asynchronously and only when needed (see pseudocode in Section 4.3). When taking parts of "good" solutions from M_A, the *information-sharing strategy* allows a maximum of $n/20$ changes so that solutions do not become too similar. When avoiding parts of "bad" solutions from M_T, the *diversification strategy* targets the disruption according to the entities that are penalized the most in each solution. In addition, the asynchronous manner in which cooperation takes place helps avoid convergence to a few too-similar solutions.

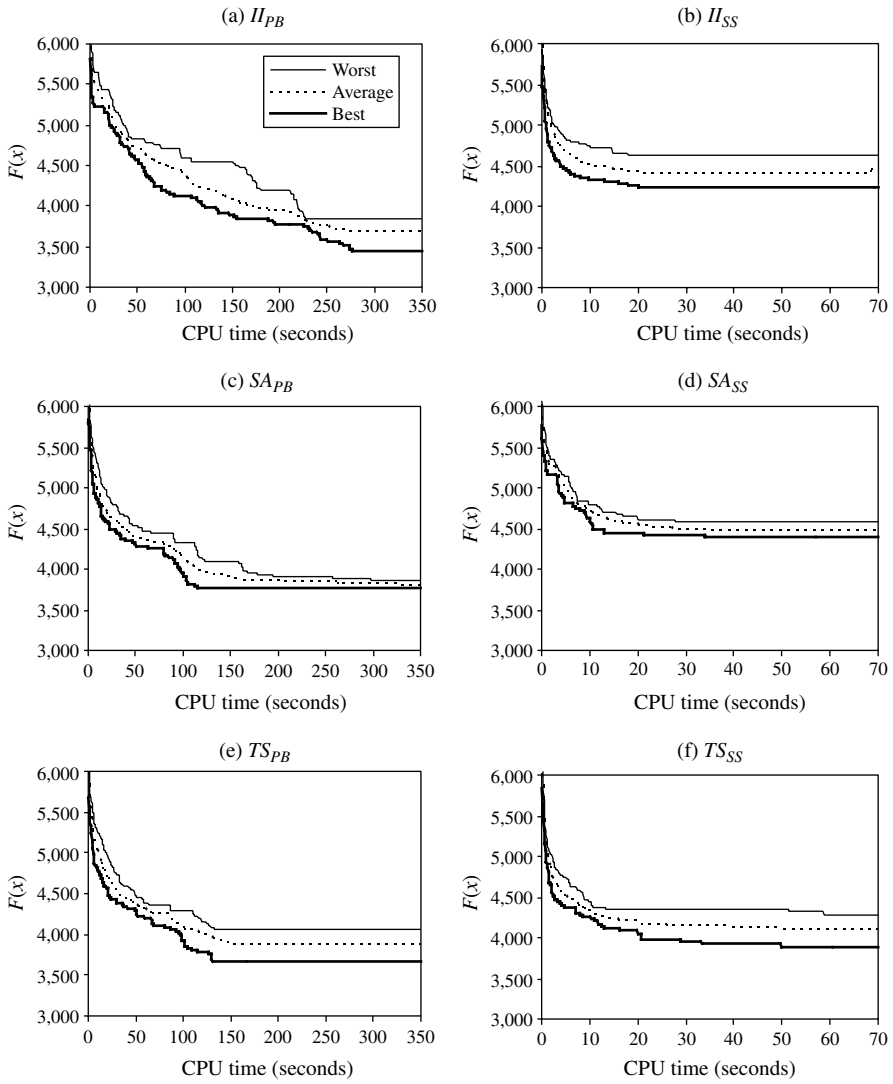

Figure 5 Rate of Improvement over Execution Time on trent1B for Each Algorithm

5.4. Rate of Improvement

The population-based variants based on our asynchronous cooperative scheme find high-quality, diverse solutions regardless of the diversity (low or high) and size (small or large) of the initial population. This section reports on the performance of the single-solution and extended methods with respect to the computation time required to achieve the results reported above. Figure 5 shows typical runs for the single-solution and extended approaches over the computation time for trent1B in which the population size is 5 and the initial population is highly diverse ($65\% < V_{ip} < 90\%$), showing the quality of the best, average, and worst individuals in the population at each time during the run. The processing time for the extended approaches is five times longer ($5t_{ind}$) than the processing time for the single-solution methods. However, as explained in Section 5.1, the total time spent by each variant to process the whole population is exactly the same. Similar results were observed for all problems in runs with different population sizes and different initial diversities.

Figure 5(b) shows that the single-solution variant of iterative improvement (II_{SS}) achieves its best performance very quickly, slightly less than 20 seconds. For the population-based variant of the same algorithm (II_{PB}), comparable high-quality solutions are found after 100 seconds, although further improvement and the best average are achieved after 250 seconds (Figure 5(a)). In other words, it takes about 275 seconds for the extended approach to find the best values for the best, average, and worst statistics in this population of 5 solutions. For the single-solution variant it takes about 20 seconds to achieve its best statistics for each of the individuals in the population. As expected, the single-solution variant is computationally cheaper but no further improvements can be achieved after a certain computation time. Although

the population-based approach takes more time to produce a set of high-quality solutions, even the worst solution outperforms the best result from the single-solution method. Moreover, after a long processing time II_{PB} still improves the average quality of the population while II_{SS} produces no better result. Assuming we need only one good solution and the computation time available is small, II_{SS} is a perfectly acceptable approach. However, if five good solutions are required to carry out comparisons and select the most appropriate one, then it will take about $20 \times 5 = 100$ seconds for II_{SS} to achieve this (by restarting the algorithm). At this time II_{PB} has already achieved a much better best solution and the average is as good (if not better) than the one produced by II_{SS}. II_{PB} can further improve the quality of the population after this computation time. Similar observations were made for the variants of simulated annealing and tabu search as illustrated in Figures 5(c)–5(f).

The execution times used here might present an advantage for the population-based approaches, so we carried out experiments running the single-solution variants for a much longer time. For example, for problem trent1B, II_{SS} was run for 350 seconds ($5t_{ind}$) for each individual in the population. That is, the total time spent to obtain the set of five solutions was 1,750 seconds. These results were compared with those obtained by II_{PB} after 350 seconds (the same as before). Even with this advantage of longer execution time, the single-solution variant was outperformed by the extended approach. The same was observed for the three algorithms in all test problems, i.e., none of the single-solution methods achieved further improvements after much longer execution times. Thus, our *asynchronous cooperative local search* approach prevents the cooperative threads from getting stuck in local optima, as appears to be the case in the independent threads.

5.5. Comparison with Prior Results

A single-solution hybrid metaheuristic (referred to as HMH_{SS}) was presented in Burke et al. (2001a) and the pseudocode is shown below. This hybrid algorithm incorporates features from iterative improvement, simulated annealing, and genetic algorithms. The mutation operator in Step 5.2 below unallocates a small number ($\approx n/10$) of entities from their assigned room. Here we extend the HMH_{SS} approach to a population-based variant (HMH_{PB}) using our asynchronous cooperative local search scheme. Note that Step 5.3 is added here when this algorithm is extended to a population-based approach.

Hybrid metaheuristic approach (HMH_{SS}) from Burke et al. (2001a)

Step 1. Generate the initial solution x using heuristic H_I

Step 2. Set $T_0 \leftarrow 1{,}000$, $T \leftarrow T_0$, $\Delta T \leftarrow 200$, $R_{iter} \leftarrow 0$, $R_{step} \leftarrow 10n$, $D_{step} \leftarrow n/2$, *iter* $\leftarrow 0$, *failed move attempts* $\leftarrow 0$, and *max failed attempts* $\leftarrow n/5$.

Step 3. For R_{step} iterations do

3.1. Find a feasible candidate solution x' using H_{LS}, and if x' is better than x then $x \leftarrow x'$.

Step 4. Find a feasible candidate solution x' using H_{LS} and if this solution is found,

4.1. Calculate the fitness variation ΔF between x and x', and if x' is better than x then $x \leftarrow x'$.

4.2. If x' is not better than x and $T = 0$,

4.2.1. Reject x', set $R_{iter} \leftarrow R_{iter} + 1$, and if ($R_{iter}$ mod R_{step}) $= 0$ then $T \leftarrow T_0$ and $R_{iter} \leftarrow 0$.

4.3. If x' is not better than x and $T > 0$,

4.3.1. *accept probability* $\leftarrow e^{-\Delta F/T}$, if *accept probability* $>$ uniform $[0, 1]$ random number then $x \leftarrow x'$.

4.4. *iter* $\leftarrow$ *iter* $+ 1$.

4.5. If (*iter* mod $D_{step} = 0$) then make $T \leftarrow T - \Delta T$.

4.6. If $T \leq 0$ then make $T \leftarrow T_0$.

Step 5. If no feasible candidate solution x' was found,

5.1. Increment *failed move attempts*.

5.2. If *failed move attempts* $>$ *max failed attempts* then apply the *mutation operator* to x.

5.3. If in cooperative scheme then use the *asynchronous cooperation mechanism*.

Step 6. Stop if the termination criterion is satisfied; otherwise go to Step 4.

Burke et al. (2001a) also presented population-based variants of HMH_{SS}, focusing on implementing a common annealing schedule to control the evolution of the whole population. By controlling the annealing schedule and the termination condition it was possible to obtain a population of high-quality solutions ($PMH - M$) or only one high-quality solution (with the rest of the population being less fit) in less time. The results produced by the $PMH - M$ method and reported in Burke et al. (2001a) are the best published results for the test instances there. Our test instances (Section 2.3) are more difficult because they are larger in terms of n and m and also incorporate more constraints (soft and hard) than those in Burke et al. (2001a).

Below we compare the performance of our asynchronous cooperative algorithms (II_{PB}, SA_{PB}, TS_{PB}, HMH_{PB}) and the $PMH - M$ algorithm from Burke et al. (2001a). The extended variant of HMH_{SS} (HMH_{PB}) differs from the $PMH - M$ approach in that HMH_{PB} incorporates our asynchronous cooperation mechanism while $PMH - M$ incorporates the common annealing schedule. See Burke et al. (2001a) for details on HMH_{SS} and $PMH - M$.

5.6. Experiments with Previous Test Instances

First, our extended approaches are tested on the instances in Burke et al. (2001a). The three problems

Table 3 Performance of the Population-Based Algorithms on the Test Instances of Burke et al. (2001a)

	P1 Idle iterations = 1,700		P2 Idle iterations= 2, 800		P3 Idle iterations = 3,500	
Algorithm	Best	Average	Best	Average	Best	Average
$PMH-M$	1,995.4	2,759.1	2,572.8	3,281.8	3,042.1	5,674.2
II_{PB}	3,256.2	4,521.8	3,017.8	3,556.3	5,484.1	6,984.3
SA_{PB}	2,147.4	2,501.8	2,305.0	3,502.7	3,820.5	5,407.9
TS_{PB}	1,742.9	2,385.4	**2,173.1**	2,947.1	2,395.1	4,401.8
HMH_{PB}	**1,512.1**	**2,193.3**	2,184.2	**2,729.4**	**1,825.5**	**3,596.9**

considered have the following main characteristics: P1 with $n = m = 55$, $h = 7$, $g = 18$; P2 with $n = 93$, $m = 55$, $h = 23$, $g = 38$; P3 with $n = m = 115$, $h = 32$, $g = 35$. The same experimental settings and computing facilities in Burke et al. (2001a) to test the $PMH-M$ algorithm were used here: a population size of $p = 10$, a number of idle iterations (iterations with no improvement) as the termination criterion, and ten runs of each algorithm on each test instance. Table 3 shows the number of idle iterations for each test instance and the results obtained with each algorithm. For each problem instance and each approach, the best and average solutions are reported. The results shown for the $PMH-M$ algorithm are from Burke et al. (2001a). The best results among all algorithms for each test instance are in bold.

The new population-based variant of the hybrid metaheuristic (HMH_{PB}) outperforms the previous extended version ($PMH-M$), i.e., the *asynchronous cooperation mechanism* helps produce better results. The population-based tabu-search approach also produces better results. In fact, the overall best solution for the P2 instance is produced by TS_{PB}, although the best average performance over all runs for all test instances is achieved by HMH_{PB}. The population-based variant of iterative improvement produces poor results. The population-based variant of simulated annealing is competitive with $PMH-M$ but produces significantly worse results than TS_{PB} or HMH_{PB}. All the population-based approaches outperformed their corresponding single-solution algorithms (as they did in the experiments of Section 5.2).

5.7. Experiments with New Test Instances

The five population-based algorithms are compared using the test instances described in Section 2.3. The two versions of the extended hybrid metaheuristic ($PMH-M$ and HMH_{PB}) were tested using the same experimental settings described in Section 5.1. Taking into account all the results obtained by each algorithm on each test instance, the overall best and average solutions are in Table 4. The results in the first three rows are those reported in Section 5.2, while the results in the last two rows are obtained in this section. Again, the best result among all the algorithms for each instance is indicated in bold.

The best solutions are also produced by HMH_{PB}, i.e., the asynchronous cooperative approach obtained from extending the single-solution hybrid metaheuristic. Again, the TS_{PB} approach is competitive and even gets one best average solution quality result (nott1c). The population-based simulated annealing algorithm (SA_{PB}) and the $PMH-M$ approach are similar on these test instances, while the population-based iterative improvement algorithm (II_{PB}) appears once again to produce less favorable results overall.

5.8. Structure of Solutions

Table 5 shows the average value (rounded up to the next integer) for each of the penalties that contribute to the overall penalty function $F(x)$. Row 3 shows the average value of space-misuse $f_1(x)$. Row 10 shows the average value of violation-of-soft-constraints $f_2(x)$. Although minimizing space overused is more important than minimizing space wasted (2), the capacity of rooms is typically

Table 4 Performance of the Population-Based Algorithms on Our Test Instances

	nott1 $p \cdot t_{ind} = 2{,}400$ s		nott1b $p \cdot t_{ind} = 1{,}200$ s		nott1c $p \cdot t_{ind} = 600$ s		trent1 $p \cdot t_{ind} = 1{,}400$ s	
Algorithm	Best	Average	Best	Average	Best	Average	Best	Average
II_{PB}	568.1	728.4	468.4	544.0	348.2	424.6	3,439.1	3,736.2
SA_{PB}	543.7	687.0	470.7	575.3	342.5	418.7	2,724.4	3,756.4
TS_{PB}	491.2	680.1	432.6	547.7	323.8	**391.4**	2,682.9	3510.6
$PMH-M$	525.9	647.7	458.0	505.8	334.9	398.5	3,217.4	3,618.7
HMH_{PB}	**482.2**	**621.5**	**417.1**	**479.5**	**315.4**	392.1	**2,531.4**	**3,104.0**

Table 5 Typical Penalty Values in the Solutions Obtained for Each Problem Instance

	nott1 $n = 158$ $m = 131$	nott1b $n = 104$ $m = 77$	nott1c $n = 94$ $m = 94$	trent1 $n = 151$ $m = 73$
Space wasted	45	17	23	43
Space overused	148	107	166	48
Space misuse	194	125	186	101
Not sharing	50	0	0	2,850
Be allocated in	40	80	100	150
Be adjacent to	40	30	70	0
Be away from	0	0	0	0
Be together with	80	50	0	100
Be grouped with	223	234	122	0
Soft constraints	538	385	264	3,280
Allocated entities	158	104	94	151
Rooms used	124–126	70–72	93–94	72–73

exceeded. This is the result of minimizing the violation of proximity/adjacency and grouping soft constraints. If a higher factor than 2 is applied to space wasted (2), more of these soft constraints are violated in the solutions produced by the algorithms. The *be grouped with* soft constraint (ensuring that entities are close to each other) is the most challenging, as illustrated by the high penalties in Table 5 for this soft constraint. The last two rows in Table 5 show the number of allocated entities and the typical number of rooms used in the solutions obtained for each problem instance. All entities are always allocated because that is a condition for feasibility. However, the number of rooms used in the solution is sometimes less than the total of rooms m, i.e., sometimes some rooms are left free, which is an interesting option for space administrators as it gives more flexibility for space management.

6. Summary and Final Remarks

We have reported results from a range of experiments on extending four single-solution techniques: iterative improvement, simulated annealing, tabu search, and a hybrid metaheuristic, toward population-based approaches based on *asynchronous cooperative local search*. The *asynchronous cooperation mechanism* consists of adding an *information-sharing strategy* and a *diversification strategy*. The *information-sharing strategy* allows individuals in the population to share good and bad parts of solutions during the search. The *diversification strategy* uses knowledge gained by the population during the search and encourages individuals to explore different areas of the solution space. Because each individual in the population uses mainly local search, no specific mechanism is required to maintain diversity (in the solution space) within the population. This is a good alternative for improving upon the performance of single-solution metaheuristics when a set of solutions is required. In producing a good set of solutions, the performance of each asynchronous cooperative approach is better than that of the corresponding single-solution algorithm. The population size and diversity in the initial population does not decrease the effectiveness of the extended variants. This is an attractive feature of the scheme because other population-based approaches such as genetic algortihms usually require larger populations to operate, or they tend to converge prematurely unless mechanisms to maintian diversity are incorporated.

Our main purpose is to illustrate the concept of *asynchronous cooperative local search* toward the design of population-based metaheuristics. We also justify the effectiveness of the method by presenting the best available results on a set of test instances of the office-space-allocation problem. This is a real-world combinatorial optimization problem with a complex search space due to the high number of hard and soft constraints. We have shown that the best results are produced by the HMH_{PB} approach and that very competitive results are obtained with the TS_{PB} approach. Future research includes applying our algorithms to real-world problems and teaching-space-allocation problems (in collaboration with Real Time Solutions (see http://www.realtimesolutions-uk.com/), to implement the *asynchronous cooperative local search* for other combinatorial problems, to investigate the systemic behavior as suggested by Toulouse et al. (2004), and to modify our approach to make it a *multi-level cooperative search* (see Crainic et al. 2005).

Acknowledgments

The authors are very grateful to Michel Gendreau and the anonymous referees for valuable comments and suggestions, which helped to improve this paper.

References

Aarts, E., J. Korst, W. Michiels. 2005. Simulated annealing. E. Burke, G. Kendall, eds. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques.* Springer, New York, 187–210.

Benjamin, C., I. Ehie, Y. Omurtag. 1992. Planning facilities at the university of Missouri-Rolla. *Interfaces* **22**(4) 95–105.

Burke, E. K., J. D. Landa Silva. 2006. The influence of the fitness evaluation method on the performance of multiobjective search algorithms *Eur. J. Oper. Res.* **169**(3) 875–897.

Burke, E. K., D. B. Varley. 1998. Space allocation: An analysis of higher education requirements. *The Practice and Theory of Automated Timetabling II: Selected Papers from the 2nd Internat. Conf. Practice and Theory of Automated Timetabling, LNCS*, Vol. 1408. Springer, New York, 20–33.

Burke, E. K., P. Cowling, J. D. Landa Silva. 2001a. Hybrid population-based metaheuristic approaches for the space allocation problem. *Proc. 2001 Congress on Evolutionary Comput.*, IEEE Press, Seoul, Korea, 232–239.

Burke, E. K., P. Cowling, J. D. Landa Silva, B. McCollum. 2001b. Three methods to automate the space allocation process in UK universities. *The Practice and Theory of Automated Timetabling III: Selected Papers from the 3rd Internat. Conf. Practice and Theory of Automated Timetabling, LNCS*, Vol. 2079. Springer, New York, 254–273.

Burke, E. K., P. Cowling, J. D. Landa Silva, S. Petrovic. 2001c. Combining hybrid metaheuristics and populations for the multiobjective optimisation of space allocation problems. *Proc. 2001 Genetic and Evolutionary Comput. Conf.*, Morgan Kaufmann Publishers, San Francisco, CA, 1252–1259.

Calegari, P., G. Coray, A. Hertz, D. Kobler, P. Kuonen. 1999. A taxonomy of evolutionary algorithms in combinatorial optimization. *J. Heuristics* **5**(2) 145–158.

Crainic, T. G. 2005. Parallel computation, co-operation, tabu search. C. Rego, A. Bahram, eds. *Metaheuristic Optimization Via Memory and Evolution, Ser. Oper. Res./Comput. Sci. Interfaces*, Chap. 13, Vol. 30. Kluwer, Boston, MA.

Crainic, T. G., M. Toulouse. 2003. Parallel strategies for metaheuristics. F. W. Glover, G. A. Kochenberger, eds. *Handbook of Metaheuristics*. Kluwer, Boston, MA, 474–513.

Crainic, T. G., M. Gendreau, J. Y. Potvin. 2005. Parallel tabu search. E. Alba, ed. *Parallel Metaheuristics: A New Class of Algorithms*, Chap. 13. Wiley, New York.

Crainic, T. G., M. Toulouse, M. Gendreau. 1997. Toward a taxonomy of parallel tabu search heuristics. *INFORMS J. Comput.* **9**(1) 61–72.

Gendreau, M., J. Y. Potvin. 2005. Tabu search. E. Burke, G. Kendall, eds. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Springer, New York, 165–186.

Giannikos, J., E. El-Darzi, P. Lees. 1995. An integer goal programming model to allocate offices to staff in an academic institution. *J. Oper. Res. Soc.* **46**(6) 713–720.

Glover, F., M. Laguna. 1997. *Tabu Search*. Kluwer, Boston, MA.

Hertz, A., D. Klober. 2000. A framework for the description of evolutionary algorithms. *Eur. J. Oper. Res.* **126**(1) 1–12.

Landa Silva, J. D. 2003. *Metaheuristics and Multiobjective Approaches for Space Allocation*. School of Computer Science and IT, University of Nottingham, UK.

Le Bouthillier, A., T. G. Crainic. 2005. A cooperative parallel metaheuristic for the vehicle routing problem with time windows. *Comput. Oper. Res.* **32**(7) 1685–1708.

Martello, S., P. Toth. 1990. *Knapsack Problems—Algorithms and Computer Implementations*. Wiley, New York.

Preux, Ph., E. G. Talbi. 1999. Towards hybrid evolutionary algorithms. *Internat. Trans. Oper. Res.* **6**(6) 557–570.

Ritzman, L., J. Bradford, R. Jacobs. 1980. A multiple objective approach to space planning for academic facilities. *Management Sci.* **25**(9) 895–906.

Talbi, E. G. 2002. A taxonomy of hybrid metaheuristics. *J. Heuristics* **8**(5) 541–564.

Toulouse, M., T. G. Crainic, B. Sanso. 2004. Systemic behavior of cooperative search algorithms. *Parallel Comput.* **30**(1) 57–79.

Verhoeven, M. G. A., E. H. L. Aarts. 1995. Parallel local search. *J. Heuristics* **1**(1) 43–65.