# Optimizing Personal Computer Configurations with Heuristic-Based Search Methods

Vincent Tam and K.T. Ma

{vtam,makengte}@comp.nus.edu.sg
Department of Computer Science
School of Computing, National University of Singapore
Lower Kent Ridge Road, Singapore 119260.

**Abstract.** Undoubtedly, the hardware technology of personal computers (PCs) are continuously changing and shaping our daily living. However, given the diversity of PC hardware components, and the limited compatibility between some of these hardware components, most people are interested to obtain an (sub-)optimal configuration for some specific usage restricted to their budget limits and other possible criteria. In this paper, we firstly formulate these widely occurred configuration problems as discrete optimization problems in which users can flexibly add in or modify their specific requirements at any time. More interestingly, we proposed two intelligent optimizers: a simple-yet-powerful beam search method and a min-conflict heuristic-based micro-genetic algorithm (MGA) to solve this real-life optimization problem. The beam search represents a restricted intelligent search while the MGA is an evolutionary algorithm with small population size. To investigate the feasibility and efficiency of our proposals, we built a Web-based Personal Computer Configuration Advisor to integrate each of the two optimizers as individual component to configure PCs for general users. In our empirical evaluation, the heuristic-based MGA consistently outperformed the beam search method in most cases. Furthermore, our work opens up numerous exciting directions for future investigation including the improvement of our optimizer to handle more complicated users' requirements, the integration of other optimizers like the branch-and-bound heuristic search method [1,2] for comparison, and the possible uses of efficient learning algorithms such as the ID3 algorithm [2] to classify different user-defined configurations into useful examples to guide the search during optimization.

**Keywords : Discrete Optimization Problems, Heuristic Search Methods, Micro-genetic Algorithms.**

## 1. INTRODUCTION

To obtain the maximal benefits out of nowadays' personal computer (PC) hardware technology, many companies or families are keen in configuring and building their own personal computers (PC) to suit their specific requirements, especially due to the influence of the popular "Do-It-Yourself" (DIY) philosophy, the fast development of multimedia hardware and the wide accessibility of the Internet. Undoubtedly, the PC configuration problems are common decision problems faced by many people in which people are always interested to know the optimal, or possibly sub-optimal, PC configuration within their limited budget. However, since the PC technology nowadays are changing very quickly, the diversity of PC hardware components, such as the different types of processors, random-access memory (RAM) and the motherboards, and their limited compatibility between certain components due to the underlying proprietary manufacturers often complicates the whole decision problem, thus making it difficult to handle by the general public.

In fact, there were many well-studied configuration problems [2] in the area of computer-aided design or manufacturing (CAD/CAM) [6]. In particular, there was an interesting research work [6] on formulating the machine configuration problems as discrete optimization problems, and then applying local search methods such as genetic algorithms [11,12] or simulated annealing [9,10] to handle these configuration problems successfully. The preliminary experimental results reported in [6] already revealed that formulating the machine, or any general, configuration problems as constrained optimization problems (COPs) would definitely result in systematic handling of users' requirements as constraints or optimization criteria. At the same time, the users can flexibly add in or modify their requirements at any time to see the resulting configurations for planning or control [5]. Thus, in this paper, we firstly gave a formal definition of these PC configuration problems as discrete optimization problems. More importantly, based on this formal definition, we proposed a systematic and flexible framework which allows the seamless integration of different intelligent optimizers for solving these difficult real-world discrete optimization problems.

Basically, a discrete optimization problem involves a set $Z$ of variables, each variable $V_i$ ($\in Z$) with a finite domain $D_i$ of discrete values, a set $C$ of constraints on some subsets of variables limiting the combination of values assigned to those involved variables, and a set of objective functions $f_j$ for minimization/maximization. The challenge is to find a globally optimal solution which minimizes/maximizes all the objective functions $f_j$ while satisfying all the constraints in $C$. Mathematically, the discrete minimization problem can be specified as follow.

$$\min \sum_{j=1}^{m} f_j \qquad \textit{subject to} \qquad \forall c \in C \quad cf(c) = 0$$

where $m$ denotes the total number of objective functions in the problem, and *cf(c) is an arbitrary function* which returns 0 when the specific constraint $c$ is satisfied. Otherwise, it returns 1. Clearly, to solve maximization problems, we can simply negate all objective function $f_j$ in the above formulation as $\min \sum_{j=1}^{m} - f_j$. In handling these optimization problems, which are always *NP-complete* [3,4], one of the frequently used heuristics is the branch-and-bound (B&B) heuristic [1,2] in which the exploration of any partial solution in a search tree will be abandoned immediately whenever the search cost of that partial solution, as represented by an arbitrary objective function, already exceeds the minimal cost for the optimal solution found so far. For instance, the B&B heuristic has been successfully applied to handle the famous traveling salesman problems [2,3] to guarantee the finding of the shortest path to transverse all the required cities in one round trip. However, in some sparsely constrained optimization problems such as the PC configuration problems which have too many feasible solutions, heuristic search methods may still be unable to return the globally optimal solution within a reasonable period of time. In these cases, users may also be willing to accept a sub-optimal solution. Thus, in this paper, we proposed two heuristic-based optimizers which are capable of returning sub-optimal solutions as a trade-off for efficiency. In the first proposal of our beam-search based optimizer, we used two major approaches to efficiently optimize the ultimate PC configurations for satisfying the users' requirements. First, similar to the branch-and-bound method [1,2], we prune off any alternative choice which already exceeds the planned budget during each search step. In addition, we include a constant threshold value *n* to monitor the size of the possible PC configurations we will consider in each search step so as to avoid the combinatorial explosion problem due to the diversity of different PC components. In our second proposal, we used an interesting min-conflict heuristic (MCH) [13] based micro-genetic algorithm (MGA) [11,12,14], that is an evolutionary algorithm with small population size (usually around *6-20* according to [11]), to solve these practical configuration problems. In fact, in our previous work [12], we already demonstrated how to successfully apply this MCH-based evolutionary approach to obtain impressive performance on solving a set of constraint satisfaction problems (CSPs). Here, our proposal represents the first attempt, up to our knowledge, to adapt this MCH-based evolutionary approach to handle real-world discrete optimization problems. Nevertheless, since both of our proposed optimizers are not performing exhaustive search, it is impossible to guarantee the global optimality of the resulting configurations. Besides, it should be interesting to note that both intelligent optimizers are heuristic-based and *restricted* in different ways for reasonable performance. The beam-search based optimizer is restricted by the number *n* of the best partial solutions considered in each search step while the MCH-based MGA is restricted by the number of candidate solutions (chromosomes) in the whole population.

To investigate the feasibility and efficiency of our proposals, we firstly collected the actual data from some major computer shopping centers in Singapore to build our local databases of PC components and then a Web-based PC Configuration Advisor as a flexible and useful platform for easy integration of different components. Based on this flexible framework, we used the widely available Practical Extraction and Report Language (PERL) [7, 8] Version 5.0 to implement the two different optimizers to compare their performance on handling these practical PC configuration problems. The heuristic-based MGA mostly outperformed the beam search method in the empirical evaluation. Obviously, our work opens up many new directions for future investigation such as applying our current optimizers to handle more complicated users' requirements, the integration of other optimizers such as the branch-and-bound heuristic search method [1,2] or SAT-related search algorithms [15] for studying their performance, and the possible uses of efficient learning algorithms such as the ID3 algorithm [1,2] to classify different user-defined configurations into useful examples to guide the search during optimization.

This paper is organised as follows. Section 2 describes the PC configuration problems as specific instances of discrete COPs, thus forming a systematic framework for optimizing choices of PC components to satisfy the users' requirement. In Section 3, we describe a flexible system architecture to handle these PC configuration problems, and detail our interesting proposals of using heuristic-based optimizers to handle these real-life COPs according to their unique problem structures. Section 4 provides the empirical evaluation of the proposed optimizer in terms of efficiency and costs of the resulting configurations, with an example application of our optimizer to build a Web-based PC Configuration Advisor. Lastly, we conclude our work in Section 5.

## 2. Personal Computer Configuration Problems

Basically, the personal computer (PC) configuration problem is to select a configuration of PC hardware parts to build a complete system, taking into account the *compatibility* issues between the different hardware components. For instance, Intel Pentium II CPU should be attached to a Slot-1 Motherboard. Definitely, one of the most important optimization criteria in many real-life situations is *price*. Thus, a general formulation of the PC configuration problems as discrete COPs can be as follows.

$$\min \sum_{j=1}^{m} \cos t(Pj) \qquad \textit{subject to} \qquad \forall comp(Pi, Pj) \in C \qquad cf(comp(Pi, Pj)) = 0$$

in which cost(*Pj*) denotes the cost for the component *Pj*, *C* specifies all the compatible relations (constraints) between the components *Pi* and *Pj*, and the arbitrary function *cf* returns *0* when *comp(Pi, Pj)* is evaluated as true. For example, when the variables for the CPU and motherboards are : $P_{CPU}$ = *"Intel PII CPU"* and $P_{MB}$ = *"Slot-1 Motherboard"*respectively, $comp(P_{CPU}, P_{MB}) = true$, then $cf(comp(P_{CPU}, P_{MB})) = 0$. Clearly, given the above general COP formulation, it is flexible to add in or modify the users' requirements specified as constraints or optimization criteria. For instance, when new components $P_X$ and $P_Y$ are added into the PC market, it is easy to simply add a new constraint *comp*$(P_X , P_Y)$ into $C$ to store their compatibility information.[1] Besides, for more complicated real-life cases, we can simply extend the minimization function to consider other important factors such as a weighting value for each component to reflect the users' preference.

**Variables**

The variables *Pi's* of interest for the PC configuration problem is the set of hardware components which will be assembled to build a functioning personal computer system. Basically, from our general analysis, there are about 12 variables commonly involved in configuring a PC nowadays. The first three variables, namely CPU, Memory and Main-Board, represent the most important variables which are often highly constrained. The remaining nine variables, including Casing, Display-Card, Hard-Disk, CD-ROM Drive, Modem, Monitor, Printer, Scanner and Sound-Card, are totally un-constrained. Thus, this special relationship between the PC components depicts the unique problem structure of the PC configuration problem: the constraints between the first 3 variables will leave very few combinations of options to be matched against a possibly large number of possible choices in a later stage of the search. It should be noted that this large possibility occurred in the later search stage may pose major difficulty to most optimizers such the branch-and-bound techniques since no other available information can be used to prune off any value during the search.

**Domains**

From the empirical data we collected, the average domain size for the first three highly constrained variables is 52 while the average domain size for the remaining un-constrained variables is also 52, thus making the average domain size for all the variables is 52. Hence, the size of the search space for all possible combinations of PC components is roughly of the order of $52^{12}$ . More importantly, this huge search space has lots of possible sub-trees which cannot be further pruned off by any heuristic. Accordingly, the conventional enumerative search [5] may simply give unsatisfactory performance to return the globally optimal PC configuration. Therefore, in the next section, we will discuss our proposed search strategies which sacrifices global optimality for efficiency to tackle these real-life COPs.

Furthermore, it should be noted that these PC, or possibly other, configuration problems are very much different from some famous routing problems such as the traveling salesman problem, on which the branch-and-bound (B&B) strategy

[1] Clearly, some readers may argue that these compatibility constraints are in fact no different from the logical relations like "father(john, mary)" to specify "john" is the father of "mary" in some logic programs. Of course, we would agree on that since it is generally true for most constraints. However, the "encapsulation" of these general relations between objects as a specific constraint will in general provide *a more systematic way* to handle that particular kind of constraints. As a result, we will discuss how a special-purpose search

performs reasonably well, in terms of the underlying domains for each variable. For a traveling salesman problem with *12* cities, the underlying domains for all the variables $V_1..V_{12}$ , denoting the first to the last city to be visited on a trip, are the same. That is the range [1..12] to represent the **same** *12* cities. Thus, at each search step, we can use the B&B heuristic to prune off any alternative route starting and ending with the same cities, say "city 1 - …..- city 9", but with a higher total distance covered. However, in our PC configuration problems, the possible choices at each search step will be totally different from the previous choices already occurred in the previous search steps. Thus, the B&B heuristic may not necessarily be a useful heuristic in solving the PC configuration problems.

**Constraints**

The following list shows some example of the constraints frequently occurred in the PC configuration problems. The first one specifies that the total cost of the configuration must be less than or equal to the budget predefined by the user. The second and third constraints are obviously about the *CPU, Memory* and *MainBoard.* The second one states that the type of the *CPU* must match with the type of the *MainBoard* while the last one specifies that the speed of the *MainBoard* and the *Memory* must be the same.

- TotalPrice <= Budget
- CPU.Type = Mainboard.Type
- Mainboard.Bus_Speed = Memory.Bus_Speed

Clearly, it is straightforward to add in more constraints as users' new requirements. As in most PC configurations, the cost and performance are definitely two major factors for consideration. However, depending on the different types of computer programs to be used by individuals, the machine performance requirement for different users are relative and often subjective measures although there are lots of performance measurement figures published by various PC vendors for advertisement. For instance, a general user who needs only word-processor to be run on his/her PC tends to consider a PC configuration with 'average' performance. But defining 'average' performance as a hard constraint for a particular application can readily be a tough question. Thus, to have more flexibility, we simply allow the users to express their performance requirements on the ultimate PC configurations possibly as some combined factors of users' preference or rating in the optimisation criteria rather than as hard constraints.

algorithm (as constraint solver) can be designed to handle those compatibility constraints *more efficiently* for our PC configuration problems in the next section.

## 3. OUR PROPOSED FRAMEWORK & SEARCH STRATEGIES

In this section, we will firstly look into a systematic framework for integrating different optimizers to handle the PC configuration problems based on the formulation of discrete optimization problems which we discussed in the previous section. Figure 1 shows our proposed systematic framework for integration of different optimizers.

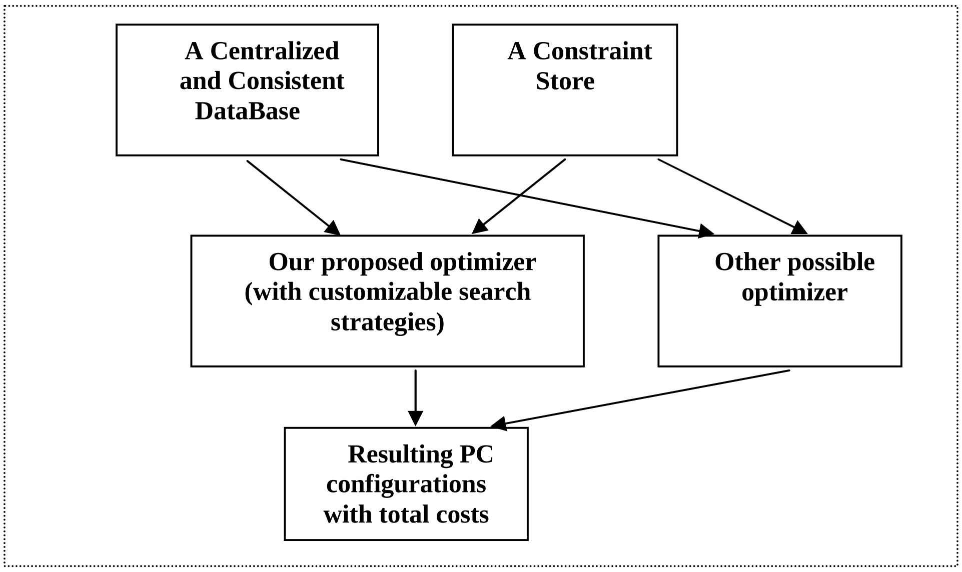

A Centralized and Consistent DataBase
A Constraint Store
Our proposed optimizer (with customizable search strategies)
Other possible optimizer
Resulting PC configurations with total costs

**Fig. 1. Our Proposed Systematic Framework for Optimizing PC Configurations**

It should be noted that in our systematic framework for constrained optimization to handle the PC configuration problems, we assume our search and optimization process have to performed on a set of consistent data from a centralized database system. In addition, there is a constraint store to provide useful information about the users' requirements as a collection of constraints. Based on these useful information, our optimizer with customizable search strategies will return a set of (sub-)optimal PC configurations with their total costs to the users according to the predefined optimization criteria. Clearly, other possible optimizers can also be added into our framework as some intelligent plug-in components for better efficiency or handling more complicated optimization criteria. Besides, in our proposed framework, we consider the optimization of options for PC configurations based on information provided from a centralized database system. It should be noted that the assumption of a centralized database system is generally valid since nowadays, most of the database systems we used in companies are often compatible with the de facto Open DataBase Connectivity (ODBC) standard [7]. In general, the heterogeneous database systems linked up via the Internet and the ODBC interface layer can be regarded a centralized database system when we ignore the latency time mainly due to the network communication and the possibly inconsistent data stored in these different database systems.

In the following, we will discuss two different intelligent optimizers with heuristic-based search strategies which may not guarantee to always return the PC configuration with the globally minimal cost. However, as opposed to enumerative search, our proposals will definitely be more efficient.

### 3.1 A Beam-Search Based Optimizer

Our first proposal of a heuristic-based optimizer for solving the PC configuration problems simply limits the search to focus on the best *n* possible values of every sorted domain to construct the partial solutions in each search step. Since the PC configuration problems are only sparsely constrained, our proposed strategy should be able to return a solution which is fairly close to the global optimal solution within a reasonable period of time. Table 3 shows the two main factors which we used to control the search, together short descriptions to explain their roles.  Both control parameters are predefined by the user of the search strategies.

| *Control* | *Short Descriptions* |
|---|---|
| **Budget** | The budget for the whole personal computer system. Total prices of the components are not allowed to exceed the budget. |
| **Threshold** | Number of the best partial solutions to be considered in each search step |

**Table 1. Our Control Factors to Search for (Sub-)Optimal PC Configuration**

Similar to the B&B heuristic, the predefined control parameter *Budget* will be used to filter out any partial configuration which already exceeds its allowed value. In addition, the *Threshold* value *n* will help to ensure the search will always return the best *n* partial configurations after sorting all possible partial configurations against their total prices at each search step. Based on these two strategies, we designed an optimizer to solve these PC configuration problems as follows.

```
1. Retrieve and sort the values for the current variable by price.
```

```
2. Find the next matching value of the variable in the sorted value queue.
Check that all the constraints are satisfied for this value. This includes the
budget constraint (that is Total_price <= Budget).

3. If no constraint is violated, goto 6.

4. Else if queue is not empty, repeat 2. Else Fail.

5. Set the current solution set to include this value.

6. If number of partial solutions < threshold, repeat 2. Else 7

7. If variable queue is not empty, proceed to set current variable to next
variable in the queue. Repeat 1.

8. If number of solutions > 0 then report Solved. Else report Failure.
```

It should be noted that our work is closely related to the 'beam search' method. In Section 4, we will give an empirical evaluation of the above-proposed optimizer to solve the actual PC configuration problems with different control parameter settings for experimentation.

### 3.2 A Min-Conflict Heuristic-Based Micro-Genetic Algorithm

Our second proposal is a min-conflict heuristic (MCH) based micro-genetic algorithm (MGA) [11,12], that is an evolutionary algorithm with small population size of chromosomes, to handle the PC configuration problems. In general, when handling constraint satisfaction or optimization problems with evolutionary algorithms, a bit-string called chromosome will usually be used to represent a possible valuation for all the variables in the constrained problem. When the valuation represented by a particular chromosome violates no constraints in the problem, the chromosome denotes a solution. For solving constrained problems with a large number of variables (say > *500*), a MGA with a small population size *PZ* in the range of *6-20* may often outperform the traditional evolutionary algorithms with larger population size [6]. This is probably because the computational cost can be minimized by focusing the search only on a reasonably small population of chromosomes without much impact on the search efficiency. Initially, the MGA randomly generates a small population of *n* chromosomes. Then, the algorithm selects the best *m* chromosomes for reproduction according to some ranking mechanism as determined by the fitness/objective value of each chromosome. There are two main types of reproductive operators: crossover and mutation to be applied on those selected chromosomes. After reproduction, a new generation is formed. And the whole process iterates until a solution is found, the whole population is converged to the same pattern or the resource limit set for the maximum number of generations (*MAX_GENS)* is exceeded.

To improve the search efficiency of MGAs in solving CSPs, researchers have tried adding different heuristics in the evolutionary computation. For instance, Rojas [17] used a heuristic to define the importance of a constraint in a constraint network to improve an evolutionary algorithm in solving a set of randomly generated 3-colouring graphs. Dozier [11,14] proposed an interesting heuristic inheritance mechanism for their micro-genetic algorithm for tackling binary CSPs. The mechanism tries to minimize the number of constraint violations by continuously mutating only a single selected variable, or moving to mutate another variable. Furthermore, they extended this micro-genetic algorithm with the integration of the Iterative Descent Method (IDM) [11], which modifies the fitness function to consider not only the number of constraint violations but to also penalize revisiting inconsistent pairs of values (no-goods) found at previous local minima during the search. Besides IDM, The min-conflict heuristic [13] forms the basis for many global and local search methods. The idea behind the min-conflict heuristic is to consider modifying only a single variable at a time, and to assign a value to that variable which is locally minimum in terms of constraint violations. When there are several local minima (ties) in terms of constraint violations, a value will randomly be selected among these ties. In our previous work [12], we proposed a mutation-based evolutionary search scheme (*MCH-MGA*) to efficiently solve CSPs as follow.

```
MCH-MGA(CSP, fitness(), PZ, MAX_GENS) {
        initialize the 1st generation of PZ chromosomes randomly;
        set best_fit = best fitness value of the current population;
        set ngens = 1;
        while (best_fit != 0 && ngens < MAX_GENS) {
          uses selection scheme to choose the pivot_gene;
          applies MCH-mutate to pivot_gene;
          applies descent-mutate to other genes;
          if (population is in local minimum) then applies popu-learn;
          update best_fit;
          ngens++;
        }
        if(best_fit == 0) return solution;
        return failure;
}
```

Clearly, the *MCH-MGA* depends on the selection scheme, either *update-select* or *usage-select*, to pick up the most important variable/gene for the current search step to apply *MCH-mutate* in order to get the greatest improvement if possible. For the remaining variables, it generally applies the more efficient operator *descent-mutate.* The *MCH-mutate* is basically a MCH variable updating operator while the *descent-mutate* operator is a restrictive operator which allows only mutations resulting in decreases in the total number of constraint violations. The *popu-learn* operator is simply an adaptation of the heuristic learning mechanism discussed in [16] to a population of chromosomes rather than a single chromosome. For detail, refer to [12].

To handle the PC configuration problems, we have to firstly modify the original *fitness* function to combine the cost of constraint violations with actual *cost* of the different PC hardware components $P_j$*'s*. Assume the average cost for each PC component $P_j$ is about *1000,* we can adjust the *fitness* function to give a relatively **larger** weight associated with the part for constraint violations as follows.

$$\min \sum_{j=1}^{m} (\cos t(Pj) + \quad 10{,}000 * \sum_{\forall comp(Pi,Pj) \in C} cf\,(comp(Pi,Pj))\,)$$

Thus, the ultimate effects produced on the evolutionary search will help to avoid any possible constraint violation. It should be noted that with this new formulation, it is straightforward to add in the users' preferences for certain PC components by adding another weighted term, for example *1000 * user_pref(Pj),* into the above *fitness* function. Besides carefully adjusting the *fitness* function, we have to modify the testing condition for the *while-loop* in the *MCH-MGA* to : *while (ngens < MAX_GENS)* since we are now handling optimization problems instead of CSPs. Therefore, the unsatisfiability testing condition: *best_fit != 0* should be removed. Furthermore, the last two statements of the *MCH-MGA* procedure should also be modified to always: return (sub-)optimal solution for the optimization problems at hands. After putting into all these minor modifications to the above *MCH-MGA* procedure, we rename this new MCH-based evolutionary algorithm for solving optimization problems as *MCH-MGA-OPT.* In the following section, we are going to compare the performance of *MCH-MGA-OPT* against the beam-search based optimizer in solving the real-life PC configuration problems.

## 4. EMPIRICAL EVALUATION

Based on the above algorithms for our proposed optimizers, namely the beam-search based optimizer and the *MCH-MGA-OPT,* we built our prototypes in PERL Version 5.0 since we would integrate our optimizers individually into a Web-based PC Configuration Advisor so as to compare their performance while facilitating the general users to configure their own PCs. To evaluate the performance of our prototypes for the proposed optimizers to handle the PC configuration problems, we focused on two main aspects of the computational results returned by our optimizer in the following analysis. The first important factor to consider is the quality of the solution in term of the total costs of the PC configurations for minimization. The second factor of interest is the efficiency of our optimizer as reflected by the timings in CPU seconds. Accordingly, we run our prototype 10 times for different settings of threshold values or combinations of parameters on a DEC-Alpha workstation running Digital Unix Version 4.0.8.

The table below shows a sorted listing of the total costs of the different PC configurations, labeled from 01 to 30, returned by our beam-search based optimizer for *Budget* <= $*3000* and *Threshold = 30.*

| Cfg Cost | Cfg Cost | Cfg Cost | Cfg Cost | Cfg Cost |
|---|---|---|---|---|
| 01 753 | 07 757 | 13 763 | 19 765 | 25 766 |
| 02 755 | 08 759 | 14 763 | 20 765 | 26 767 |
| 03 755 | 09 760 | 15 763 | 21 765 | 27 767 |
| 04 755 | 10 762 | 16 764 | 22 765 | 28 768 |
| 05 757 | 11 762 | 17 764 | 23 765 | 29 769 |
| 06 757 | 12 763 | 18 764 | 24 766 | 30 769 |

Clearly, the optimal cost returned by the beam-search based optimizer is SG $753. And there is a fair difference of SG $16 between the first minimal total cost and the last total cost of the configuration 30 as shown in the above table. In addition, it should be noted that from our initial experiments, the top 10 configuration-cost pairs usually remain unchanged for the same budget limit even though we changed the threshold value from 30 to 20 and then to 10. This demonstrated the stable performance of our beam-search based optimizer in handling these PC configuration problems.

Table 4 shows the performance of our proposed beam-search based optimizer for handling the PC configuration problems with *Budget* <= $*3000* and varying the *Threshold* value at *1, 5, 10, 20* and *30.* For each case, the reported data is the average CPU time in seconds over *10* runs.

| *Threshold* | *Average CPU time in seconds (10 runs)* |
|---|---|
| *1* | *3.18* |
| *5* | *3.19* |
| *10* | *3.32* |
| *20* | *3.40* |

|  |  |
|---|---|
| *30* | *3.51* |

**Table 2. Performance of our Beam-Search Based Optimizer for Different Threshold Values**

The above table clearly showed that our beam-search based optimizer is fairly efficient in handling a typical PC configuration problem with different threshold values to control the search. Also, the performance of this optimizer is compact and stable since there is only slight increase in the average CPU time from 3.18 to 3.51 seconds when the corresponding threshold value is changed from *1* to *30*.

Table 3 gives the performance of the *MCH-MGA-OPT* for *PZ = 1, 5, 10 and 20*[2] and *MAX_GENS = 20, 50 and 100* for comparison. Again, all the data reported are averages over *10* runs. For each table entry, the first data represents the CPU time in seconds while the second one in parentheses denotes the cost in SG$. When we compare against Table 1 for the quality of solutions as returned by the beam-search based optimizer, it is clear that the *MCH-MGA-OPT* consistently outperformed the beam-search based optimizer with the optimal cost = SG $715 for most cases. In fact, as for *MCH-MGA-OPT,* the minimal cost over *10 runs* consistently stays at SG $715 for all the cases, which probably represents the global optimal solution[3]. In general, the beam-search based optimizer could only better the average cost returned by *MCH-MGA-OPT* in the weakest case where *PZ=1 and MAX_GEN = 20.* On the other hand, the *MCH-MGA-OPT* generally outperformed the beam-search based optimizer in terms of solution quality.

| PZ\MAX_GEN | 20 | 50 | 100 |
|---|---|---|---|
| **1** | 0.35s (847.2) | 0.71s (715) | 1.28s (715) |
| **5** | 1.26s (716.8) | 3.10s (715) | 6.04s (715) |
| **10** | 2.38s (715) | 5.99s (715) | 12.03s (715) |
| **20** | 4.76s (715) | 11.97s (715) | 24.07s (715) |

**Table 3. Performance of our *MCH-MGA-OPT* for Different Population Sizes and Resource Limits**

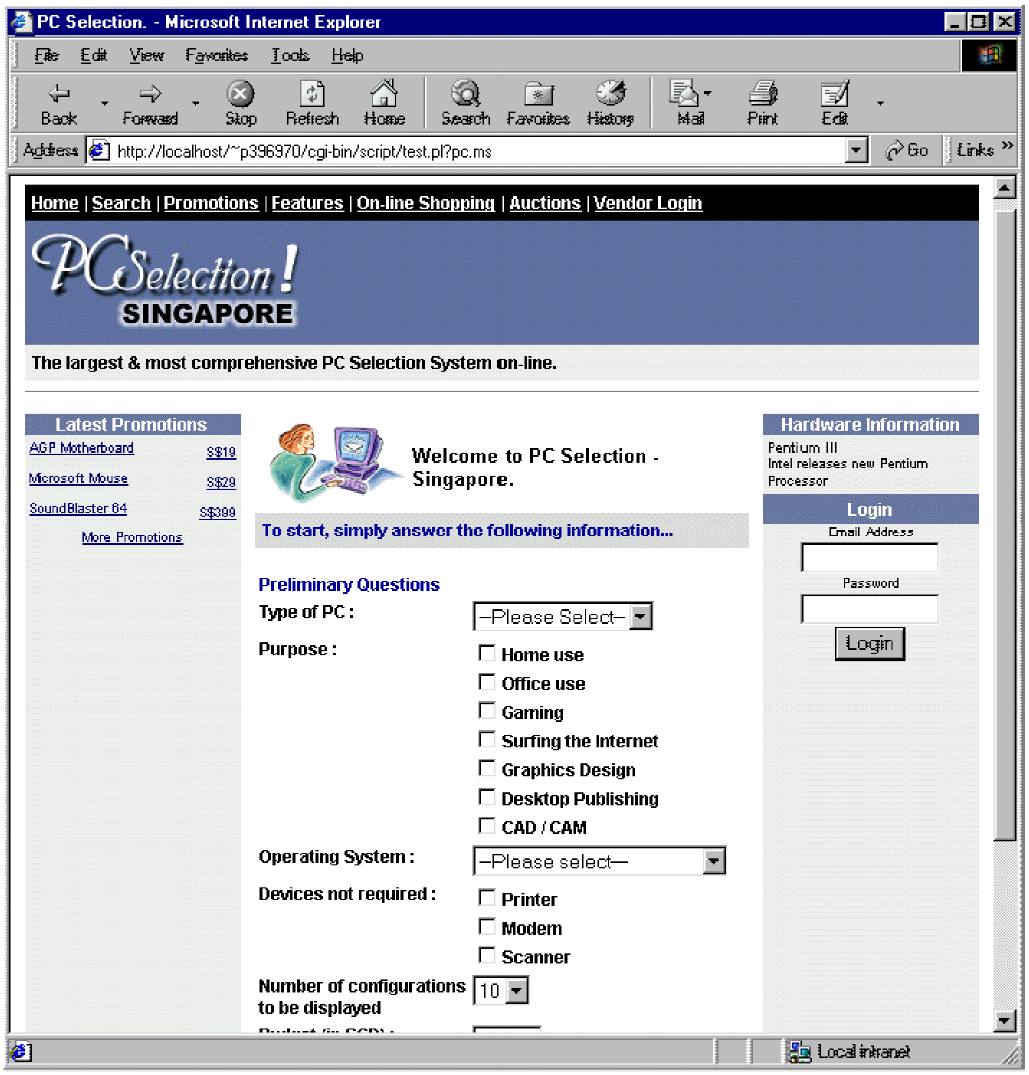
As for the CPU time in seconds, it is interesting to note that, the CPU time in Table 3 almost increases linearly across the table. For instance, when the *MAX_GEN = 20,* the CPU time (2.38s) for *PZ = 10* is almost double of that (1.26s) of *PZ = 5.* Again, this general trend is fairly consistent across the whole table. Besides, when we compare the CPU time of *MCH-MGA-OPT* against those of the beam-search based optimizer shown in Table 2, it is clear that the efficiency of the

[2] It should be noted that due to time limitation, we do not have sufficient time to obtain the data for PZ = 30. However, in the revised version, we would definitely provide this data for a more conclusive comparison.

*MCH-MGA-OPT* is comparable or sometimes even better than those results reported for the beam-search based optimizer. This is especially true for cases like *PZ = 1* for all the resource limits, *PZ = 5* for *MAX_GEN = 20* or *50* and *(PZ = 10, MAX_GEN = 20)*. Accordingly, we can safely conclude that *MCH-MGA-OPT* achieved comparable or slightly better efficiency to those results reported for the beam-search based optimizer on half of the test cases we have compared. After all, when we consider both efficiency and quality of solution, *MCH-MGA-OPT* should provide more exciting preliminary results as compared to the beam-search based optimizer. In addition, it should be noted that it is straightforward to add in more users' requirements under the evolutionary search framework.

Since the performance of our proposed optimizers from the initial experiments is satisfactory, we integrated our optimizers as a plug-in components into our targeted Web-based PC Configuration Advisor for optimizing different PC configurations according to users' budget as shown in Figure 2.

[3] Of course, this will require a complete solver to later verify the global optimality of our solution found.


PC Selection. - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Refresh Home Search Favorites History Mail Print Edit

Address http://localhost/~p396970/cgi-bin/script/test.pl?pc.ms

Home | Search | Promotions | Features | On-line Shopping | Auctions | Vendor Login

PCSelection! SINGAPORE

The largest & most comprehensive PC Selection System on-line.

**Latest Promotions**

| | |
|---|---|
| AGP Motherboard | S$19 |
| Microsoft Mouse | S$29 |
| SoundBlaster 64 | S$399 |

More Promotions

Welcome to PC Selection - Singapore.

To start, simply answer the following information...

**Preliminary Questions**

| | |
|---|---|
| Type of PC : | --Please Select-- |
| Purpose : | ☐ Home use<br>☐ Office use<br>☐ Gaming<br>☐ Surfing the Internet<br>☐ Graphics Design<br>☐ Desktop Publishing<br>☐ CAD / CAM |
| Operating System : | --Please select-- |
| Devices not required : | ☐ Printer<br>☐ Modem<br>☐ Scanner |
| Number of configurations to be displayed | 10 |

**Hardware Information**

Pentium III
Intel releases new Pentium Processor

**Login**

Email Address

Password

Login

Local intranet

Fig. 2. **Our PC Configuration Advisor**

There are a number of interesting points about our prototype system of the Web-based PC Configuration Advisor. Firstly, our Web-based PC Configuration Advisor supports rule-based information processing to actively anticipate or validate users' inputs, and also analyze hardware information stored in databases to provide useful advice on the best PC configurations for certain usage subject to a user-defined budget limit, and a pre-defined threshold limit used during the search. Specially, we allows the active uses of rules, as stated in the rule script to represent some domain experts' knowledge, to "dynamically" prune off options within the same page (intra-page) or between the pages (inter-page). For example, when the user selects the "Type of PC" to be configured is "Server" on the first page, the option of "17-inch monitor" will be automatically removed in the last page[4] with the active use of the domain experts' rules. Besides ensuring valid inputs from the users, it is mainly used to remove irrelevant choices so as to facilitate the search efficiency. After all possible pruning, our proposed optimizers, implemented as a easy-to-modify plug-in components inside our PC Configuration Advisor, then performs either the controlled (*best-n)* search according to the predefined budget limit and threshold values, or the heuristic evolutionary search according to the preset population size *PZ* and resource limit *MAX_GENS* during optimization of PC configurations. Thus, when applying the beam-search based optimizer, for a user's requirement with *budget* $\leq$ *3000 and threshold = 20,* our Web-based PC Configuration Advisor will definitely return only the first *20* optimal PC configurations with the total cost less than $*3000*. Similarly, for the evolutionary approach, the PC Configuration Advisor can also return the best *PZ* (sub-)optimal solutions as represented by all the chromosomes in the last generations. In case where no PC configuration can meet the specified budget, an error message will be printed on the resulting Web page for both cases.

## 5. CONCLUSION

It is undeniable that the PC configuration problems are both practical and interesting optimization problems widely occurring in many well-developed countries of the world. However, specific configuration problems are interesting since they have the unique problem structures that some components are highly constrained while the rest are totally unconstrained. This may pose challenge to some conventional enumerative search methods. As the PC technology is likely to be more complicated in the future, it will definitely become more difficult to handle these specific configuration problems. In this paper, we gave a formal definition of the PC configuration problems as discrete optimization problems. Based on this problem formulation, we proposed a systematic framework which allows the integration of different optimizers for optimizing the PC configurations according to the useful information stored in a centralized database and constraint store. More importantly, after analyzing the unique features of the PC configuration problems which may possibly lead to combinatorial explosion during the search, we proposed two useful search strategies to control the search and optimization process in our optimizer. To demonstrate the feasibility of our proposals, we implemented a prototype for each of our proposed optimizers in Perl for quick experimentation, and later integrated it into a Web-based PC Configuration Advisor to facilitate the general public in configuring their own PCs. In our empirical evaluation, the

[4] Technically, this special feature is achieved with JavaScript through the uses of cookies to remember some important values for each

heuristic-based evolutionary approach generally gave more promising results as compared to the beam-search based optimizer.

Clearly, there are several interesting directions for our future investigation. One obvious direction is to test our optimizers on many different real-life cases. Along this direction, it would be interesting to improve our optimizers to handle more complicated constraints and optimization criteria so as to solve more complex real-life PC, or other general, configuration problems. Second, it should be interesting to include other optimizers such as the B&B method for a complete comparison. Lastly, we are currently studying the possible uses of efficient learning algorithms such as the ID3 algorithm [2] to classify different user-defined configurations into useful examples so as to guide the search during the optimization process.

## REFERENCES

[1] "Artificial Intelligence : A Knowledge-Based Approach" by Morris W. Firebaugh, PWS-Kent Publishing Company, Boston, 1988.

[2] "Artificial Intelligence" by Elaine Rich and Kevin Knight, McGraw-Hill International Edition, 1991.

[3] "Discrete Mathematics – A Unified Approach" by Stephen A. Wiitala, McGraw-Hill International Edition, 1987.

[4] "Introduction to Algorithm" by Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest, The MIT Press, 1994.

[5] "Foundations of Constraint Satisfaction" by Edward Tsang, Academic Press, 1993.

[6] "Genetic algorithms versus simulated annealing : Satisfaction of large sets of algebraic mechanical design constraints" by A.C. Thornton, in *Proceedings of Artificial Intelligence in Design*, pp. 381-398, 1994.

[7] "Discover PERL 5" by Naba Barkakati, IDG Books WorldWide Inc., 1997.

[8] "Programming in PERL" by Larry Wall, O'Reilly, 1995.

[9] "Boltzmann machines for traveling salesman problems" by E. Aarts and J. Korst, *European Journal of Operational Research*, 39:79-95, 1989.

[10] "Optimization by simulated annealing : an experimental evaluation; Part II, graph coloring and number partitioning" by D. Johnson, C. Aragon, L. McGeoch, and C. Schevon. Operations Research, 39(3)378-406, 1991.

[11] "Solving small and large scale constraint satisfaction problems using a heuristic-based microgenetic algorithm" by G.Dozier, J. Bowen and D. Bahler. *In Proceedings of the IEEE International Conference on Evolutionary Computation*, 1994.

[12] "Improving Evolutionary Algorithms for Efficient Constraint Satisfaction" by Vincent Tam and Peter Stuckey, *International Journal on Artificial Intelligence Tools*, Vol. 8, No. 2, World Scientific Publishers, December 1999.

already visited or currently visiting page.

[13] "Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problem." S. Minton, M. Johnston, A. Philips, and P. Laird. *Artificial Intelligence*, 58:161-205, 1992.

[14] "Solving Constraint satisfaction Problems Using A Genetic/ Systematic Search Hybrid that Realizes when to Quit." Bowen James and Gary Dozier. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 122-129, 1995.

[15] "Domain-independent extension to GSAT: Solving large structured satisfiability problems." Bart Selman and Henry Kautz. In *Proceedings of IJCAI-93*, 1993.

[16] "GENET: A connectionist architecture for solving constraint satisfaction problems by iterative improvement." Andrew Davenport, E.P.K. Tsang, C.J. Wang, and K. Zhu. In *Proceedings of AAAI'94*, 1994.

[17] "From Quasi-Solution to Solution: An Evolutionary Algorithm to Solve CSP." Maria Cristina Riff Rojas. In *Proceedings of Principles and Practice of Constraint* Programming (CP96), pages 367-381, 1996.