برای پیاده سازی الگوریتم جستجوی کامل برای این مساله، کافیست پس از اضافه کردن عناصر به محموعه powerset شرایط مسئله را روی آن‌ها بررسی کنیم و جواب مسئله را پیدا کنیم.

Listing 1: Python Knapsack Problem Implementation

```
def find_most_valuable_subset(elements, W):
    p = [[]]
    subset = []
    subset_value = 0
    for element in elements:
        for i in range(len(p)):
            new_subset = p[i] + [element]
            p.append(new_subset)

            if sum([x[1] for x in new_subset]) <= W:
                new_subset_value = sum([x[0] for x in new_subset])
                if new_subset_value > subset_value:
                    subset_value = new_subset_value
                    subset = new_subset

    return subset, subset_value
```

در اینجا پس از ساخت هر زیرمجموعه، با بررسی شرایط مسئله، در صورت با ارزش تر بودن زیرمجموعه جدید آن را انتخاب کرده و در آخر بهترین زیرمجموعه به همراه ارزش آن بازگردانده می‌شود. (عناصر باید به صورت دوتایی های ارزش و وزن به تابع داده شوند). این تابع با ارزش‌ترین زیرمجموعه مجموعه‌هایی که تعداد اعضای آن‌ها تا ۲۵ باشد را کمتر از یک دقیقه محاسبه می‌کند

با استفاده از توابع آماده در پایتون برای تولید زیرمجموعه‌ها که با زبان C نوشته شده‌اند می‌توان این عدد را تا ۲۶ یا ۲۷ نیز افزایش داد. البته این روش مزیت استفاده بهینه از حافظه را به علت استفاده از generator ها را نیز داراست :

Listing 2: Python Faster Knapsack Problem Implementation

```
from itertools import chain, combinations

def faster_powerset(elements):
    s = list(elements)
    return chain.from_iterable(combinations(s, r) for r in range(len(s)+1))

def faster_most_valuable_subset(elements, W):
    best_subset = []
    best_subset_value = 0
    for subset in faster_powerset(elements):
        if sum([x[1] for x in subset]) <= W:
            subset_value = sum([x[0] for x in subset])
            if subset_value > best_subset_value:
                best_subset_value = subset_value
                best_subset = subset
    return best_subset, best_subset_value
```